Introduction of an Advanced Caching Layer Leveraging the Varnish Technology Stack and Integrating It to the Existing Web Platform

Author: Irina Vasilieva

Director: Jaime M. Delgado Merce

June 21, 2018

Barcelona School of Informatics
Universitat Politècnica de Catalunya
Thesis presented for the Bachelor Degree of Computer Science

Abstract

Web performance nowadays plays a significant role for many leading enterprises and the ones that trying to gain more visibility and users. Multiple studies and research papers in the area show that poor performance have a negative impact on business goals. An endless waiting for slow Web pages to load frustrates demanding visitors into seeking alternatives and as a result, abandoning a website.

Many different solutions and approaches came up recently to solve performance and scalability issues on the web sites. It is no surprise, that companies attempt to retain their users, improve user experience, conversions and SEO rankings to get a profitable revenue. HTTP requests reduction, file compression, infrastructure, image optimization are some of the web performance optimization techniques, or even a combination of them, enterprises find useful for their web platforms.

Varnish, an open source software, was suggested as a proxy caching server to prove that it can drastically improve hit rate and response times on the website. It can deal with performance and scalability at the highest level. In order to demonstrate the caching capability of Varnish a web platform was built based on Adobe Experience Manager with its own out of the box caching tool, named dispatcher. The main focus is to replace dispatcher and compare the web performance outcome.

Contents

Chapter 1

Context and Scope of the Project

1.1 Context and Problem Formulation

Web application performance and scalability have become a key in digital marketing business. Whatever the size of a business, Web caching can help optimize performance, save bandwidth and be as much scalable as any business may require in the future. The right caching solution can help a business to grow without the need for expensive and time-consuming re-structuring [1].

Performance is a web application's ability to execute at an acceptable level within a given time-span. But here, the performance is a relative term, since an 'acceptable level' depends on the nature of a web application and its users. This leads to a scalability, in another words it is keeping the performance stable when the load increases.

Poor performance does not only hurt the business' Google ranking, it will also impact the bottom line which is represented by the customers: people do not have the patience to wait for slow content and will look for an alternative in a heartbeat. In a heavily saturated market, they will probably end up with one of the competitors.

When building a website the content management platform is selected accurately so the editors are able to manage all of the marketing content and assets in a professional and easy way, and then deliver them to the right person at the right time. Adobe Experience Manager offers a web-based client-server system for building, managing and deploying commercial websites and related services, which can be accessible in a single platform. It combines a number of infrastructure and application level functions into a single integrated package. [2]

Caching is an essential part of any AEM project. For this reason, Adobe has always made available a dispatcher, which is a module for a HTTP server that works as a caching and/or load balancing tool. The dispatcher caches pages rendered by AEM so they are delivered by the httpd directly instead of being rendered at each request. This reduces the amount of load in AEM instances while giving the user access to a set of features provided by the httpd. The most common use of Dispatcher is to cache responses from an AEM publish instance to increase the responsiveness and security of an externally facing published website. Most of the discussion focuses on this case, but it can also be used to increase the responsiveness of an author instance, particularly if there is a large number of editors managing the content of the website. [3]

More recently, Varnish has appeared as an alternative to the dispatcher. They might look like very similar at first glance, but in reality they are very different tools with divergent objectives. Varnish Cache is now a relevant product for almost all type of business. It is designed for modern hardware, modern operating systems and modern work loads. Varnish is more than a reverse HTTP proxy that caches content to speed up the server in question, it is an HTTP accelerator or a web accelerator that is installed in front of any web or application server, which caches files or fragments of files in memory that are used to drastically reduce the response time and network bandwidth consumption on future, and equivalent requests by caching the server's output. Varnish is usually bound by the speed of the network, effectively turning performance into a non-issue, which allows the end user to focus on how the web applications work, therefore to care less about performance and scalability. [4] Varnish offers many other solutions for any size of project, but this research will focus on the caching feature mainly.

As a solution for scalability AEM suggests to use CDN (e.g. Amazon CloudFront, Akamai), to accelerate usage of AEM application and content from publish instance in a secure way. By utilizing the Content Delivery Network, developers and system administrators stand to benefit from greater simplicity associated with running Varnish, because it caches the content from the origin, but also has the ability to scale throughput and serving speed. As a result, working in combination they can deliver high performance results and fulfill the expectations.

The aim of the project is to solve the performance and scalability issues that big companies might experience on their digital platform based on AEM. And this problem can be solved by replacing the out of the box caching tool of AEM, called dispatcher, with Varnish Cache integrating it into Amazon CloudFront Content Delivery Network. As for the conclusion, the performance tests and results will be presented to finish the document.

1.2 Stakeholders

This document is primarily of interest to organizations already using Adobe AEM or planning to deploy the application in the future and want to ensure that they are getting maximum performance from their AEM environment.

1.2.1 Project Developer

Project developer handles all project development activities from initial idea, feasibility studies, requirements definition, installation and ongoing maintenance, that focus on moving toward a successful completion. This person is responsible for a successful accomplishment of the project by identifying and connecting next actions until the final objectives are reached.

1.2.2 Executives and Managers

These are mainly experts and people holding important roles in the company to take such a decision of making significant changes, who are interested in a caching solution for their website. They have a good understanding of technical vocabulary and technological stack, and need more information on how to perform certain tasks and see what level of the technological stack would be affected and in what way, so their needs are fulfilled. Sometimes, these people are only looking for a theoretical aspects on the consequences of tasks that will be performed in the future.

1.2.3 Technical People

This type of people is interested in practical aspects of how to perform certain tasks and what is the final outcome of each of them. More frequently, technical people are looking for a visual information, represented by guidelines with the followed steps or summarized results. The technicians are very well familiarized with the main topic and technology used in the project and may seek some background information to gain some knowledge and to better accomplish their challenges.

1.2.4 Beneficiaries

The main beneficiaries are the enterprises that are looking for cutting hosting related costs in terms of performance and scalability. These could be the companies interested in a better caching solution whether their websites are based on AEM or not, and also the companies that are already using AEM software and

would like to have a replacement for their current caching system. Companies websites that benefit the most from an optimized version of Varnish Cache are the content-heavy, dynamic sites that have a large number of concurrent users and spikes in traffic.

1.3 Current Solutions

There are many other solutions, as the chart 1.1 shows, to increase the speed of page load, whether it is done on the browser or server side, using a software or other optimization technologies. The best lessons are learned through experience from successful stories that offer insights and practical solutions from a wide variety of companies and industries. [5]

Cloudflare Website Optimization is a direct competitor of the Varnish Cache software and is ranked #1 out of top 12 most used Web Accelerators technologies worldwide, according to Datanyze Universe, a company that provides real-time insights based on an enterprises' technology choices and buying signals.[6] Cloudflare users can choose any combination of the web content optimization features such as Argo Smart Routing, Cache Header Optimization, Automatic Content Caching, Local Storage Caching and much more, that significantly improves performance.[7]

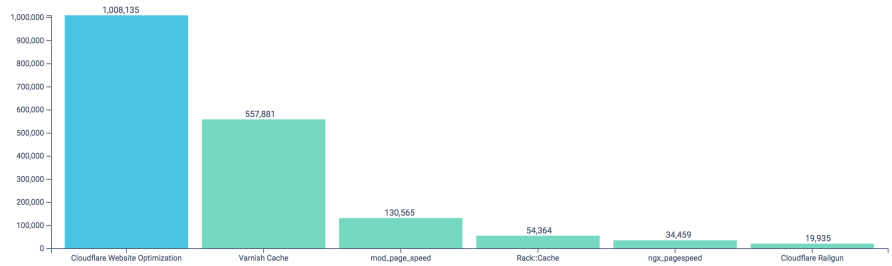


Figure 1.1: Bar Chart for Cloudfare and Its Competitors

On the contrary, one of Varnish Software's very first customers Norwegian online newspaper Vg.no was experiencing challenges with scalability and speed. Before implementing an optimized Varnish installation their objective was to make their website faster and to save costs on their web infrastructure. They attained their goal and managed to reduce their hardware costs by 90% and they saw a dramatic reduction in response times. [8]

Wetpaint.com is another example of a Varnish customer that signed up for a subscription and with help from the Varnish support team saw a significant improvement in their web performance. In November 2012 they implemented an optimized version of Varnish Cache and their site went from being unresponsive

and slow to becoming very fast. According to Tony Flint, Wetpaint's Senior QA, IT and Operations Manager, Wetpaint's average load time went from 8-9 seconds to less than 3 seconds. And after setting up Varnish they were also able to cut the number of servers supporting their site from 10 to 4. [9]

Below other testimonials follow as proof of successfully optimized websites with Varnish Cache:

> 'It's been a pleasure working with Varnish API & Web Acceleration and VCS. It is not often that one can say that about technology. In fact, it has become an essential part of our toolkit.'
> *RTÈ, Irland's National Public Service Broadcaster*

> 'Varnish is so flexible that we have been able to tune it perfectly to fit our specific needs. It is a major part of our website operations.'
> *Boozt Fashion, Swedish e-commerce leader*

> 'With Varnish API & Web Acceleration we can deliver a high performance user experience for Nikon and their customers. It's lightning fast, and flexible.'
> *Nikon, world leader in digital and photo imaging*

> 'Varnish has given us the capacity to grow without compromising on delivering the most premium user experience available.'
> *Surfline, company focused on providing up-to-the-minute ocean weather information and streaming coastal HD cam network*

> 'The well-known and proven capabilities of Varnish API & Web Acceleration established a solid ground on top of which the access control layer or paywall could be built.'
> *RCS MediaGroup and Allenta, one of the world's major multimedia publishing groups*

> 'Implementation was very simple. The platform is very versatile when making changes and adapting it to SSO, web services, etc. Unlike in other applications where everything is more proprietary and closed.'
> *La Nación, leading Argentinian newspaper*

1.4 Scope

In order to have the entire technology stack up and running, the first step will be getting a publishing and authoring instances with AEM application installed on one Amazon server for each, means there will be two dedicated servers one for authoring and one for publishing purposes. Additionally, a basic configuration of the servers should be performed, that is enabling publishing system and connecting it to the author, and activating the testing content afterwards.

Next step would be installing Varnish on the publishing environment and set it up correctly in order to get the requests from the end user cached in memory. Varnish Cache can also be installed on authoring system, so it leverages the performance for the editors, who manage the content on the platform, hence that is not the goal of the project.

Afterwards, a CloudFront distribution will be created and associated to the web server, where Varnish will reside. CloudFront is an Amazon CDN service that distributes the content of the platform globally and it is used with the main purpose of making the latency and response times equivalent for any location.

In the final phase of the project a performance tests will be done and their corresponding results will be presented and analyzed accordingly. The expectation for the final step is to achieve a significant reduction in response times when retrieving a page with its corresponding types of files.

One of the main obstacles in the given project is the domain specific language called VCL, that represents a configuration system of the software. Varnish translates VCL into binary code which is then executed when requests arrive. The VCL files are organized into subroutines, which can be modified to change the behaviour of the whole mechanism. The different subroutines are executed at different times. One is executed when the request arrives, another when files are fetched from the backend server and so on. In order to implement new caching policies for Varnish Cache this language should be first studied with the help of the corresponding material. [4]

On the other hand, implementation phase is another challenge along with the time that determines how far this project can get in terms of the level of complexity of Varnish configuration. Once Varnish is installed, it already delivers a notable performance improvements to the website, but there are many other challenges that could be resolved and this is a collection of modules extending Varnish VCL used for describing HTTP request/response policies with additional capabilities, called 'VMODs'.

1.5 Methodology

1.5.1 Agile Methodology

Applying Agile methodology to the project will help to split the challenge into small pieces and deliver the desired result in incremental and iterative work sequences until the main goal is achieved. The following figure 1.2 shows the main stages the project will follow.

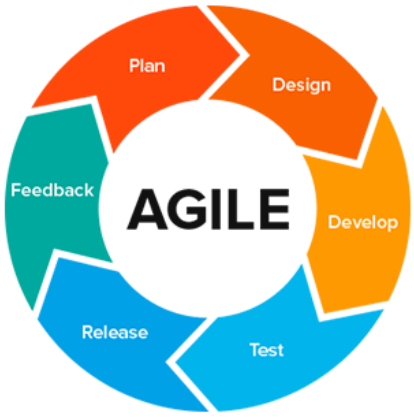


Figure 1.2: Agile Project Lifecycle

1. Plan
   Outlining a plan that would fulfill the project's needs in order to create and deliver working software as soon as possible is the main objective of the first step. The main tasks, which have been identified earlier, are prioritized and put in work.
2. Design
   The initial phase of the project is about designing and planning of the entire infrastructure to be suitable and feasible in order to fulfill the project's needs. In fact, one of the critical requirements of the project is to build an operational environment on a trustful platform that ensures stability, reliability and high availability during the service engagement, which is why the technology stack is built using Amazon Web Services connected to the Amazon CloudFront Content Delivery Network.
3. Development
   During this phase the problem is being solved by applying the configuration management process, which helps to refine and polish the final result, and improve the caching solution in place as best as possible.

4. Testing
   Completing through testing and analysis of results before delivery of the project will help to monitor the progress and again prioritize again some tasks to put more effort in those.

5. Release
   Release stage is about presenting a fully-functional solution to stakeholders and customers.

6. Feedback
   The feedback is expected to be provided by the director of the project, so he can validate the overall development process and obtained results meet the expectations.

1.5.2 Configuration Management Process

In order to accomplish the delivery of the project a configuration management process will be followed for Varnish Cache mainly, using VCL domain specific Varnish language to edit configuration files. The purpose of the Configuration Management process is to ensure that the assets required to deliver services are properly controlled, and that accurate and reliable information about those assets is available when and where it is needed.[10] This information includes details of how the assets have been configured and the relationships between them. The figure 1.3 illustrates the process flow.

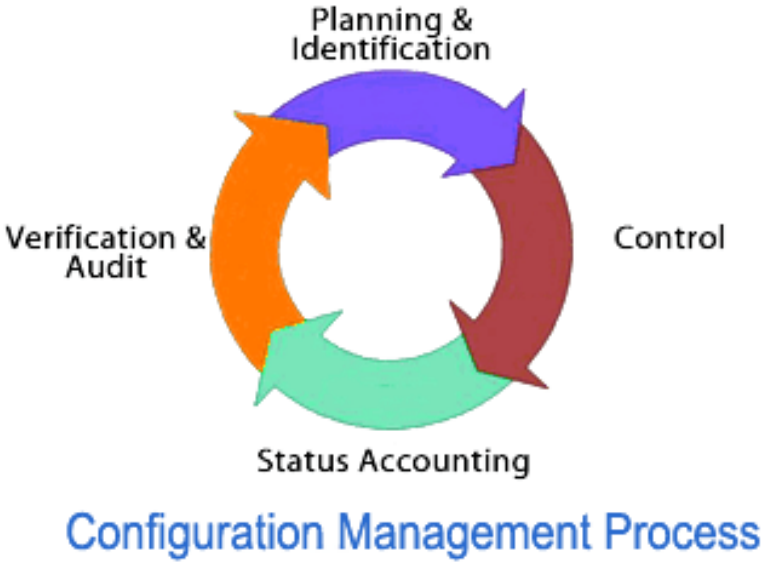


Figure 1.3: Configuration Management Process

1. Planning and Identification
   A configuration management plan describes a specific procedures and the extent of their application during the lifecycle in order to be effective, predictable and repeatable. This involves breaking down the work into component deliverables, called configuration items, creating a unique numbering or referencing system and establishing configuration baselines.
2. Control
   This step ensures that all changes to configuration items are documented. An important aspect is the ability to identify the interrelationships between configuration items.
3. Status Accounting
   This tracks the current status of a configuration, providing traceability of configuration items throughout their development and operation.
4. Verification and Audit
   This is used to determine whether a deliverable conforms to its requirements and configuration information. Typically, an audit is undertaken at the end of a phase.

1.6 Validation Method

Once Varnish installation is complete it is available and working. The application's behaviour can be easily audited by issuing Varnishtop command, as it will allow to see the most common executions of certain tags. Another useful tool to observe how Varnish is working on background is Varnishlog and it is used to access request-specific data, providing large amounts of data about specific clients and requests, this way the validation will be done quickly as Varnishlog would start recording logs transactions to memory.

Chapter 2

Project Planning

2.1 Planning and Organization

The estimated duration of the project is of 4 months, that starts in the end of February and ends in the end of June, right before the final presentations, by setting up the technology stack and running the tests afterwards to get the final results. However, there is an iterative part within the development phase that follows the configuration management process and allows to modify the existing features or add a new ones, which are not covered in the planning, but still could be included in the final project.

2.2 Task Description

Since all the former parts of the stack are identified and ready to be integrated the development process, including the resources, both material and human, for each of them follow next.

2.2.1 Local simulation

This process consists of creating a local environment similar to the described in the project, excluding the CDN, get some knowledge on how all software components work and interact with each other. The table 2.1 describes the required resources:

Human Resources	System Engineer, Solution Architect, DevOps should be involved in order to set the local environment, verify the correctness of the technology stack and plan alternative solutions beforehand.
Hardware Resources	MacBookPro 2,8GHz CPU with at least 4 cores 16GB of RAM minimum 64-bit Operating System 128 Gb disk or more
Software Resources	Vagrant, a virtual machine manager to create 2 instances of AEM

Table 2.1: Required Resources for Local Simulation

2.2.2 Amazon Web Services

As a first step an appropriate environment will be configured, one Authoring and one Publishing instance which will help to publish and access the content. At this point, there is no content but the required instances are acquired. Amazon Web Services offer Free Tier that includes offers that expire 12 months following sign up and others that never expire, and they perfectly solve the needs of the project. Two different servers, as specified in table 2.2 will be used and this way the Authoring system will be separated from the Publishing one.

Human Resources	System Engineer, Solution Architect, DevOps should be involved in the initial phase of setting up the environments, monitoring tools and corresponding alerts if apply.
Hardware Resources	Amazon S3, EC2 Intel Xeon or AMD Opteron CPU with at least 4 cores 16GB of RAM minimum 64-bit Operating System 128 Gb disk or more
Software Resources	According to technical requirements for AEM [11] and Varnish [12], authoring system are not differentiated form the publishing one, therefore the characteristics will be the same.

Table 2.2: Required Resources for AWS

Adobe Experience Manager

Regarding the AEM application that will be running on both Authoring and Publishing environments, the license of the product should be purchased. Gartner, the world's leading research and advisory company that issues a Magic Quadrant every year to position technology players within a specific market, named Adobe's WCM offering as one of the more expensive in the market in 2015, maintaining its position nowadays as well, sometimes being twice the price of its nearest competitor. [11] The licensing fees for Adobe Experience Manager and Adobe Marketing Cloud largely depend on the business and which components are implemented and the costs are near $250,000 to $1,000,000 and up annually. Hence, the company also offers a 90-days trial version, which can be acquired by contacting Adobe Sales team. Once the trial version license is facilitated the installation of the application will not take long. The table 2.3 represents the needed resources for this step.

Human Resources	System Engineer, Solution Architect, DevOps should be involved in order to set the needed parameters within the application, such as level of logs, alerts, memory and CPU usage.
Hardware Resources	5 GB free disk space in the installation directory 2 GB of RAM 5.4 GB of temporary space
Software Resources	Oracle SE 8 JDK 1.8.x (64bit) Linux, based on Red Hat distribution, CentOS Apache httpd 2.4.x Web Browser - most of them are supported

Table 2.3: Required Resources for AEM

Apache Web Server

Reaching this point, the Varnish Cache software is already installed and might be running on the system, despite that it will not be caching any single request until some adjustments are made on Apache side. As the figure 2.1 suggests, the reverse proxy Varnish should be put in front of the web service to accelerate responses to HTTP requests and reduce server workload. Varnish works by handling requests before they reach the backend; whether this backend is Apache, nginx, or any other web server. If it does not have a request cached, it will forward the request to the backend and then cache its output. This way Varnish is able to store these cached requests in memory, so they are retrieved by and delivered to clients much faster than they would be from disk. [12]

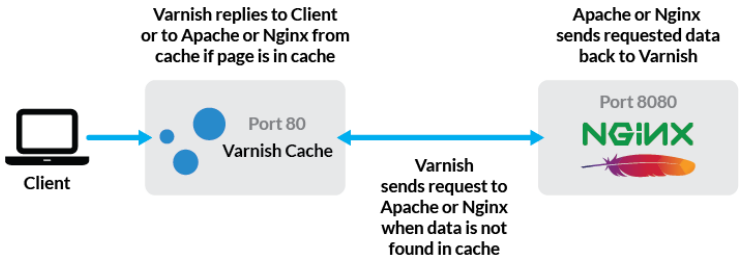


Figure 2.1: HTTP Traffic Flow

The table 2.4 describes the main requirements for Apache:

Human Resources	System Engineer, Solution Architect, DevOps should take care of the mentioned application, configure and adjust the parameters when it is needed.
Hardware Resources	Apache should be installed on AWS servers
Software Resources	APR and APR-Util Perl-Compatible Regular Expressions Library (PCRE) Disk Space at least 50 MB of temporary free disk space available ANSI-C Compiler and Build System

Table 2.4: Required Resources for Apache

2.2.3 Dispatcher

Dispatcher is an out of the box AEM caching tool, therefore it requires the same resources as AEM as the table 2.5 shows. At this point the caching tool is ready to be adjusted in order to start caching the content.

Human Resources	System Engineer, Solution Architect, DevOps should be involved to adjust the needed parameters and configure the dispatcher properly so it is able to cache the content.
Hardware Resources	5 GB free disk space in the installation directory 2 GB of RAM 5.4 GB of temporary space
Software Resources	AEM pre-installed Linux, based on Red Hat distribution, CentOS Apache httpd 2.4.x Web Browser - most of them are supported

Table 2.5: Required Resources for Dispatcher

2.2.4 Varnish

As soon as AEM is installed, next task will be initiated and it is about Varnish installation and configuration. It should be mainly on the Publishing system since it contains the content exposed directly to the end-user, that will be cached by Varnish. The developer has to get familiar with the VCL domain specific language first, as this is the proper way to get the Varnish Cache software to work according to the project requirements. Every inbound request flows through Varnish and the only way to alter this behaviour is by editing the VCL code. Varnish Cache is an open-source software, meaning it can be downloaded from its official website and after that run the installation instructions step by step, always paying special attention to the developer packages and libraries dependencies. The table 2.6 represents the resources Varnish will require.

Human Resources	System Engineer, Solution Architect, DevOps should be involved to adjust the configuration of the application, set alerts and have full control.
Hardware Resources	RedHat / CentOS el6 or el7 64-bit Operating System 4GB of RAM
Software Resources	Apache httpd 2.4.x

Table 2.6: Required Resources for Varnish

2.2.5 Amazon CloudFront

Amazon CloudFront will be used as CDN to distribute globally the content and improve the response times of delivered content to the end users. The table 2.7 explains the required resources for this phase.

Human Resources	System Engineer, Solution Architect, DevOps should take care of the mentioned application, configure and adjust the parameters when and where it is needed.
Hardware Resources	AWS instances launched with public IPs.
Software Resources	Varnish Cache installed earlier.

Table 2.7: Required Resources for CDN

2.2.6 Performance Tests

Finally, speaking about testing for the proposed solution, Apache provides its own testing tool, called Apache Benchmark. It is a tool for benchmarking Apache Hypertext Transfer Protocol (HTTP) servers. It is designed to give an impression of how the current Apache installation performs, especially to show how many requests per second the web server is capable of serving. [13] To ensure optimal performance from a Varnish Cache, Apache Benchmark will be used during the Configuration Management Process in the following scenarios:

- when installing a Varnish Cache
- when applying VMODs to Varnish Cache
- when writing caching policies in the Varnish Configuration Language

The table 2.8 explains main requirements to perform the final tests.

Human Resources	System Engineer, Solution Architect, DevOps should take care of the mentioned step, since he/she was involved in the whole process and from now on should start collecting the results and analyze them in order to keep optimizing the system.
Hardware Resources	The tests are performed on the AWS servers.
Software Resources	The tests are performed using a common browser and/or console on the AWS instances, mainly on Publisher side.

Table 2.8: Required Resources for Testing

2.3 Monitoring and Control

2.3.1 Estimated Timetable

An estimated number of hours is shown in table 2.9:

Task	Estimated time(h)
Local Environment Simulation	80
Validation	20
AWS Instances Setup	7
Install Apache	3
Install AEM	3
Install Varnish	3
Adjust Apache and Varnish	3
Register Public Domain	3
CloudFront Service Setup	5
Connect AWS Servers to CloudFront	3
Get Familiar with VCL	40
Performance Tests	60
Iterative Part of Configuration	150
Generate Documentation and Presentation	20
Total	400

Table 2.9: Timetable

2.3.2 Gantt Chart

An estimated scheduled of tasks is shown in table 2.2

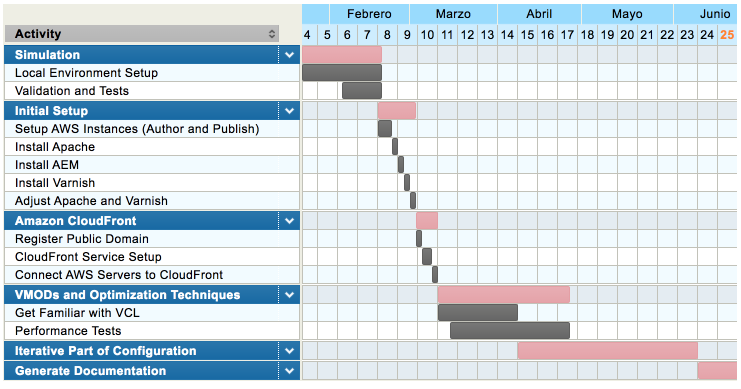


Figure 2.2: Gantt Project Planner

2.4 Alternatives and Action Plan

2.4.1 Time Consuming Tasks

Within the configuration management process there could be some varnish modules [14] and optimization techniques, that will take more time than planned. At the same time, there could be some of them that would require more time than others. As a consequence, the half implemented or not implemented optimizations will not be tested and therefore not documented. The reason for that, is that it is hard to calculate how much time is needed to implement a certain technique optimization or tweak a certain vmod. As a solution for that would be to choose the most important or relevant ones for the scope of the project and order them by time consuming.

2.4.2 Bugs and Package Dependencies

Another, not covered by the estimated time, issue is any kind of possible bugs or library dependencies. During the development phase of the project the develop packages and libraries used by Varnish, Apache and AEM may change. As a result, an issue may occur blocking the progress of the tasks. In order to prevent this kind of issues, every single change in the systems should be documented, the systems in their turn should be monitored with the help of Amazon tools and application-level logs.

2.4.3 Administrative Issues

Last but not least, the administrative issues may be presented during the project evolution. Due to the fact, that the trial versions of all products vary a bit and do not overlap in time, the developer should stick to the most limited one, so they are all available at the same time until the project is fully complete. In case of trial versions, the developer will have to make sure the product is still available until the final date of the presentation, otherwise a local environment with the simulated tasks should be used. With regards to Amazon Web Services, the number of permitted hours per month and per instance should not be overpassed so the servers are fully operational, otherwise an additional price will be charged.

Chapter 3

State of the Art

3.1 Amazon Web Services (AWS)

Since 2006, Amazon Web Services has been the world's most popular and broadly adopted cloud platform. AWS offers on-demand cloud computing platforms to individuals, companies and governments, on a paid subscription basis over 100 fully featured services for compute, storage, networking, database, analytics, application services, deployment, management, development, mobile, Internet of Things (IoT), Artificial Intelligence (AI), security, hybrid and enterprise applications, from 55 availability zones within 18 geographic regions round the world. AWS services are trusted by millions of active customers around the world — including the fastest-growing startups, largest enterprises, and leading government agencies — to power their infrastructure, make them more agile, and lower costs. [15]

3.1.1 AWS Overview

Elastic Compute Cloud (EC2)

Amazon Elastic Compute Cloud provides scalable computing capacity based on the users needs and choice of main requirements such as hardware, CPU, RAM, disk space and OS, so the user can develop and deploy applications faster without any upfront commitment. The end user is allowed to configure and customize on the mentioned virtual or dedicated servers security settings, networking, manage storage, or even scale up or down to handle changes in requirements or spikes in popularity, reducing the need to forecast traffic. [16]

Elastic IP

An Elastic IP address is a static IP address, available in v4 only, designed for dynamic cloud computing, which is reachable from the Internet. It can be easily associated with the EC2 instances to enable communication with the visitors. Even if the machine is temporarily stopped, the IP is maintained associated until it is explicitly removed detached from it. [17]

Identity and Access Management (IAM)

Access to the AWS resources can be restricted using the IAM. This service offers many different ways of managing the access control including inbound traffic, security groups, roles and permissions policies, so the application running on the machine is protected not only on the http server side, but also on AWS platform.

Virtual Private Cloud (VPC)

VPC hosts the services on a private network which is not accessible from the Internet, but can communicate with the resources in the same network. This restricts the access to the resources for those who should not be allowed to access.

3.2 Dispatcher

Dispatcher is Adobe Experience Manager's caching and load balancing tool and works as an HTTP server module in front of the AEM Publishing instances. Its most commonly used feature is the caching, helping to store as much of the static website content as possible making infrequent the access to the website's content management system for the end users. [18]

3.2.1 Dispatcher Overview

As shown on the figure 3.1 an advanced layout engine processes the request from an end user by reading content from a repository and transforming the content into a document that is tailored to the visitor's rights, according to the permissions and roles set in AEM. However, the layout engine requires more processing power than a static server, so this setup may slowdown if many visitors use the system. In order to accelerate the delivering the dispatcher module renders documents and stores them in the filesystem, so the HTTP server can deliver them as if they were a simple static files. [3]

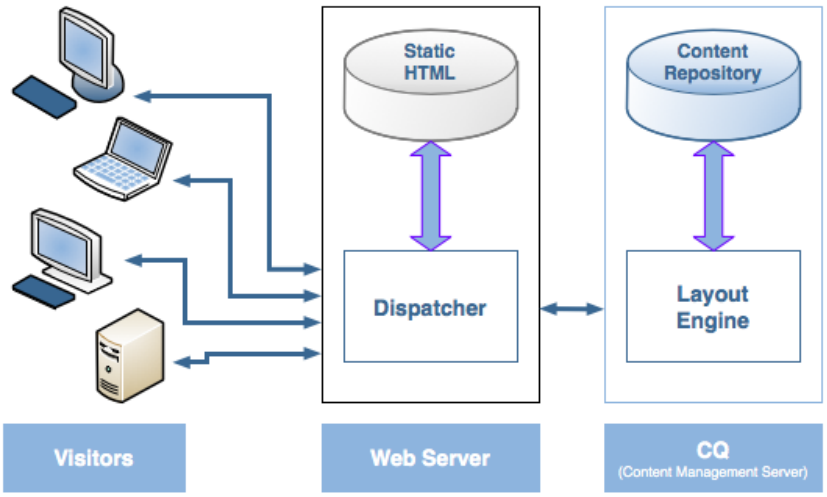


Figure 3.1: How Dispatcher Performs Caching

When modifications are made on the Authoring system the dispatcher acts in a two different ways to update the cache content, explained below:

- Content Updates
  It removes the pages that have been changed, as well as files that are directly associated with them, such as media assets and styling.
- Auto-Invalidation
  It automatically invalidates those parts of the cache that may be out of date after an update by replacing stale objects by the new ones.

3.3 Varnish

Varnish is a reverse HTTP proxy, also known as a web application accelerator. A reverse proxy is a proxy server that appears to clients as an ordinary server installed in front of the web server with the main purpose of caching the incoming requests and delivering them faster on future. It acts transparently for the visitors in order to speed up the response times and help to handle high traffic delivering dynamic and heavy content. Due to the low cost and impressive results of the software with the 13% of the top 10,000 websites many well-known brands such as The New York Times, The Guardian, Facebook, Twitter, Reddit, Vimeo among others rely on Varnish.

3.3.1 Varnish Overview

The main challenge of Varnish Cache has always been performance and scalability. It was designed to speed up the response times and improve traffic rates on modern hardware and to run on 64-bit architecture. Mainly, because it allocates more memory and more number of threads, which allows Varnish to perform at the highest level.

Varnish uses a workspace-oriented memory-model instead of allocating the exact amount of space it needs at run-time. Nevertheless, it does not manage its allocated memory, instead it takes advantage of the kernel' skills. By doing this, Varnish can move a lot of the complexity into the OS kernel, which is better positioned to decide which requests and responses are ready to handle and when. As a consequence, Varnish liberates itself from useless and resource consuming tasks and focuses on others.

Varnish uses the Varnish Configuration Language (VCL) that allows to configure and adjust the default features according to the web application's needs. VCL is translated to C programming language code. This code is compiled with a standard C compiler and then dynamically linked directly into Varnish at run-time. Next section explains the VCL in more detail.

3.3.2 Varnish Configuration Language (VCL)

Varnish has implemented state machines to process client and backend requests efficiently. Each state represents a certain C function of the Varnish core code and it is called to process the request or response. For most of the states, core code also calls into a state-specific function, called a VCL subroutine.

There are 12 subroutines, explained in tables 3.1 and 3.2 presented below, that control how Varnish behaves and can be changed in order to improve hit rate and speed up the content delivery of the website. [19] The graphs, following the subroutines explanation, attempt to provide an overview over the processing states, their transitions and the most relevant functions in core code. [20]

Client Side Subroutines

`vcl_recv()`
This is the first state of the Varnish workflow once a request is received from the browser. Its aim is to decide whether or not to serve the request, possibly modify it and decide on how to process it further.
`vcl_pipe()`
Called upon entering pipe mode from the previous state, where no other VCL subroutine will ever get called after it. In this mode, the request is passed directly to the backend without caring about caching.
`vcl_hit()`
Called after a cache lookup when the object requested has been found in the cache. The object being hit may be stale, meaning the Time-ToLive(TTL) may be equal zero or a negative value, but it still will be delivered by Varnish and not requested to the web server.
`vcl_miss()`
This state is called after a cache lookup when the object requested was not found in the cache with the main purpose to decide, based on the default or customized configuration file, whether or not to retrieve the document from the backend.
`vcl_hash()`
The aim of this state is to create a hash key for a new request and store it for future hits. This hash value will be used to look up the object in Varnish.
`vcl_purge()`
This subroutine is called to execute the cache invalidation in Varnish and therefore to flush the objects with all their possible variants.
`vcl_deliver()`
This state is called to deliver the object whether it is sent from from the cache or the web server.
`vcl_synth()`
This function is used to generate content within Varnish such as personalized error messages and 301/302 redirects among others. This content is generated in VCL and coming from backend.

Table 3.1: Client Side Varnish Subroutines

Backend Side Subroutines

`vcl_backend_fetch()`
Once this state is called, the request may be abandoned by returning a synthetic error generated in VLC or delivering the object form the backend, previously caching it or not.
`vcl_backend_response()`
This function is called from the previous state because the request could be fetched in the backend and there is still a request to respond, whether delivering it from the backend, abandon with an error or pass without taking any action.
`vcl_backend_error()`
This subroutine is called when the backend fetch failed or if maximum number of retries to finish the backend transaction has been exceeded.

Table 3.2: Back Side Varnish Subroutines

The diagram 3.2 represents the client side workflow of the subroutines. [21]

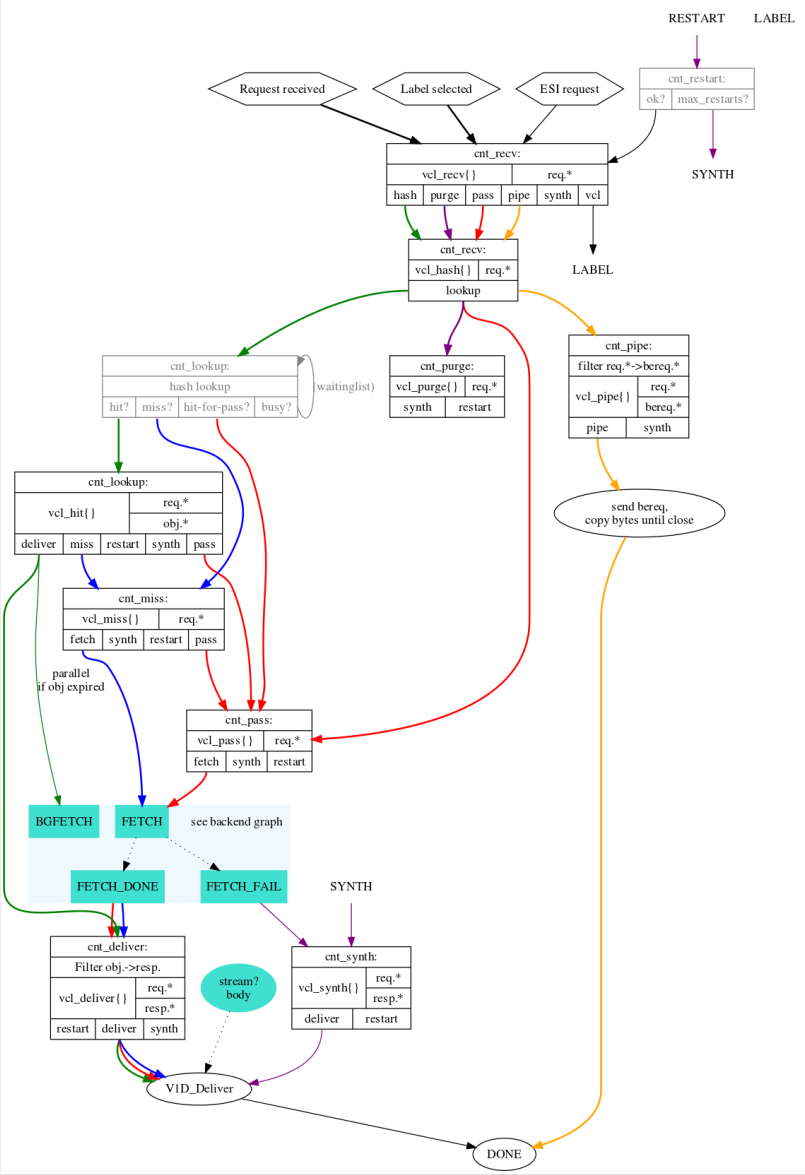


Figure 3.2: Client Side

The figure 3.3 represents the backend side workflow of the subroutines. [21]

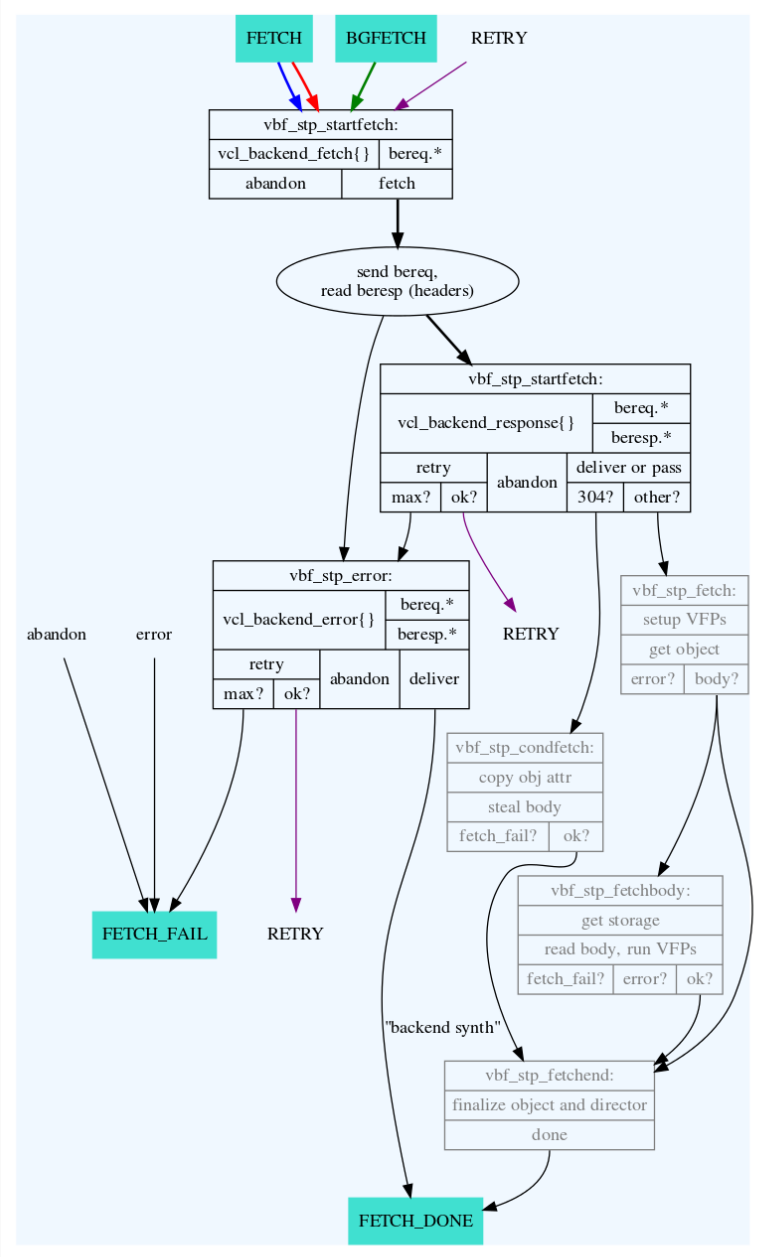


Figure 3.3: Backend Side

The diagram 3.4 represents the client side workflow of the subroutines simplified by Section.io[22]

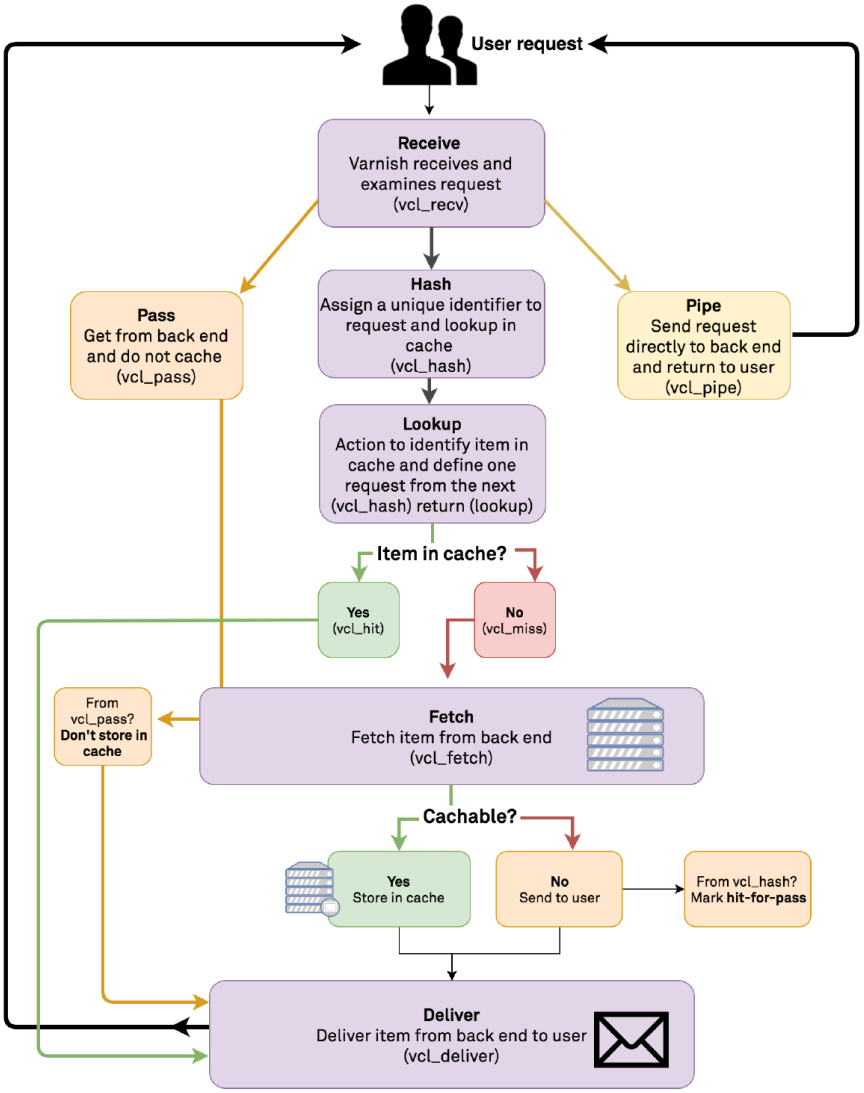


Figure 3.4: How Varnish Performs Caching

3.4 Amazon CloudFront CDN

Amazon CloudFront is a global content delivery network (CDN) service that securely delivers data with low latency and high transfer speeds. It is deeply integrated with key AWS services, including physical locations that are directly connected to the AWS global infrastructure. CloudFront offers a simple, pay-as-you-go pricing model as well as all Amazon web services. [23]

3.4.1 Route 53

In order to use CDN service a domain is needed, which can be registered in a so called Route53. It is a highly available and scalable Domain Name System (DNS) service that effectively connects user requests to infrastructure running in AWS – such as an Amazon Elastic Compute Cloud (Amazon EC2) instances, but can also be used to route users to infrastructure outside of AWS.

The following figure 3.5 illustrates a complete technology stack, where the CloudFront sits in front of Varnish, distributing the content globally, reducing the latency and response times independently of the geographical localization.

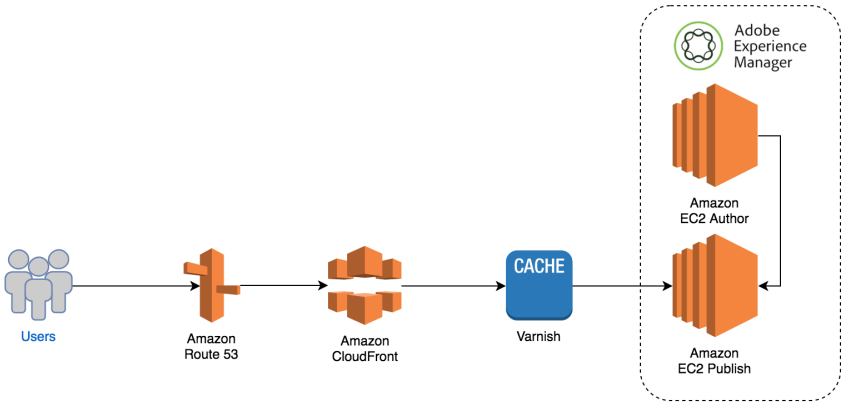


Figure 3.5: CloudFront in Technology Stack

Chapter 4

Development

4.1 Local Simulation

The local simulation is the initial stage of the Development phase within the Agile methodology. It is as important and advantageous as the development phase itself due to the benefit it provides before the project developer starts working with the planned tools. The reason to that, is that during this stage some issues may arise and the developer than should be able to find a quick solution and alternatives in order to point the progress of the project in the right direction.

Install and Configure Software Components

Generally speaking, the installation of the software components achieved satisfactory results, including some library dependencies and updates on the local environment, which also were expected.

Vagrant

The local environment was based on CentOS using Vagrant virtual box manager to start and stop the instances easily. As per requirements, two boxes with the AEM application were needed, one for publishing purposes and another one for authoring purposes. The following script 36 - Vagrantfile shows how Vagrant initializes boxes based on the pre-configured parameters such as IPs, ports and local resources.

```
Vagrant.configure("2") do |config|

  config.vm.box = "centos/7"
  config.vm.define(:author)  { |author|
    author.vm.hostname = "my-local-author"
    author.vm.network "private_network", ip: "
       33.33.33.10"
    author.vm.synced_folder "~/.m2", "/home/vagrant/.
       m2"

    author.vm.provider "virtualbox" do |vb|
          vb.customize ["modifyvm", :id,
        "--name", "my-local-author",
        "--memory", "8192", "--cpus", "2",
      ]
    end

    author.vm.network "forwarded_port", guest: 80,
       host: 48080, auto_correct: true
    author.vm.network "forwarded_port", guest: 444,
       host: 48443, auto_correct: true
  }

  config.vm.define(:publish)  { |publish|
    publish.vm.hostname = "my-local-publish"
    publish.vm.network "private_network", ip: "
       33.33.33.11"
    publish.vm.synced_folder "~/.m2", "/home/vagrant/.
       m2"

    publish.vm.provider "virtualbox" do |vb|
          vb.customize ["modifyvm", :id,
        "--name", "my-local-publish",
        "--memory", "8192", "--cpus", "2",
      ]
    end

    publish.vm.network "forwarded_port", guest: 80,
       host: 49080, auto_correct: true
    publish.vm.network "forwarded_port", guest: 443,
       host: 49443, auto_correct: true
  }
 end
```

Listing 4.1: Vagrant Script

AEM

Since AEM is a java application, the only requirement for the Vagrant box was to install JDK package and start AEM on the correct port, which is mentioned in the Development section. No modifications were made on the application level, as it is installed along with the testing content that can be used once started.

Dispatcher

Similarly to AEM, since the dispatcher is the AEM out of the box tool, there was no difficulty to install and start dispatcher on the machine. The configuration applied to dispatcher and mentioned in the Development section was tested first on the local machine.

Varnish

The caching proxy is easy and fast to install. However, its difficulty is hidden in the amount of available tools and Varnish Configuration Language(VCL). So, to start adjusting and adapting Varnish to the website's nature, tools such as varnishadm, varnishstat and varnishlog where studied and analyzed the outcome.

Akamai Varnish Connector

The Akamai Connector for Varnish was planned to be used along with the Akamai CDN. The Akamai Connector for Varnish is a VMOD that transparently syncs Varnish cache configuration and purges to the Akamai platform using VCL (Varnish Configuration Language). Since it is a VMOD written in the same language as Varnish, it needs to be installed and compiled. So, at this point, the only action to do was to attempt to install and avoid any possible issues that might have arisen. Once installed, the Connector would automatically sync with the Akamai Platform based on the Varnish configuration and forward the cache purges, making Varnish the source of truth for cache storage. However, Akamai platform has been replaced by the AWS CloudFront. [24]

Research and Learning

A wide range of technical articles, documentation, reviews, configuration guidelines, including the "Getting Started with Varnish Cache" book were read and some exercises based on Varnish Book[21] were completed in order to gain knowledge around Varnish software and develop the project in the correct context. A part from the written support material, the project developer also made use of the webinars and multiple video trainings found on the official platforms, such as AWS and Varnish Software.

4.2 Amazon Web Services (AWS)

4.2.1 Reasons to Build on AWS

Cost-Effective

AWS offers a pay-as-you-go approach for pricing for any service within AWS Cloud, meaning that the end user is charged for the individual services only for as long as they are in use, and without requiring long-term contracts or complex licensing. And once the services are not in use, there are no additional costs or termination fees. The costs can be adjusted on the go for single components such as CPU, storage, RAM or services such as management or development tools.

Free Tier

AWS offers a Free Tier with no cost associated, which provides enough credit to run an EC2 micro instance 24/7 all month. The overall duration of the trial period is of 12 months and it enables to gain free hands-on experience with the AWS Cloud services. In order to gain some experience and knowledge around AWS platform this service was used in the project as part of the development phase before building the technology stack on the final servers.

Performance

AWS delivers High Availability Performance providing a wide range of services across multiple regions in order to be closer to the customers or to meet the legal requirements, according to the business needs and expectations. The web applications can be protected from any failures that might affect certain region by launching them in separate zones, which are called Availability Zones, according to AWS.

Security

According to the "AWS Security Best Practices", AWS provides a global secure infrastructure and foundation compute, storage, networking and database services, as well as higher level services. It also provides a range of security services and features that AWS customers can use to secure their assets, that helps them to fulfill their responsibility for protecting the confidentiality, integrity, and availability of their data in the cloud, and for meeting specific business requirements for information protection.

Flexibility

One of the most important AWS features is the capacity to get a running a server in an instance, able to receive requests and answer them, and quickly shut down instances when they are no longer needed.

Reliability

Most of the Amazon Web Services meet 99.99% of commitment, making themselves reliable. Amazon's virtual backbone that has been honed for over a decade makes the user confident to build on it.

Scalability

Using AWS tools, Auto Scaling, and Elastic Load Balancing, web applications can scale up or down based on demand. Amazon's massive infrastructure compute and storage resources, with full access as well, whenever needed.

4.2.2 Server Setup EC2

In order to launch a new EC2 instance several steps are needed to correctly set up a machine according to the application requirements and resource consumption. The following figure 4.1 illustrates the EC2 Dashboard where all the instances and related data is managed.

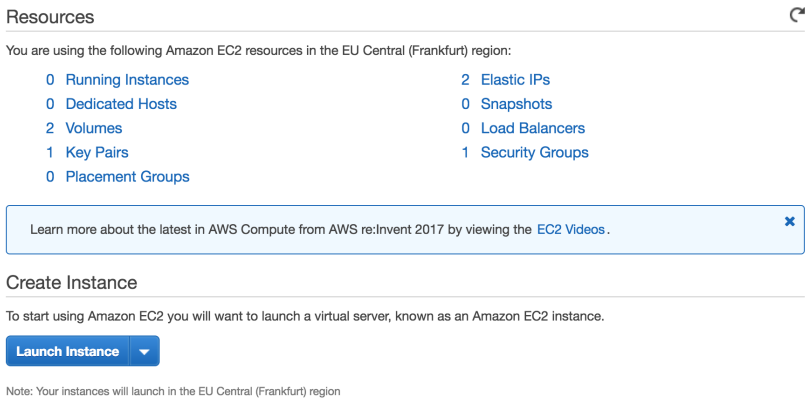


Figure 4.1: Launch New Instance

Once the election of the EC2 instance is initiated, in the first step a new machine can be selected by Operating System, architecture, root device type, pre-built packages and software needed of a certain purpose, as the illustration 4.2 shows.

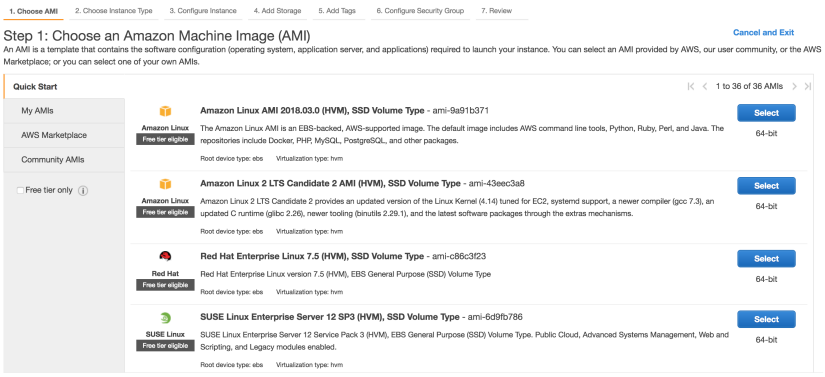


Figure 4.2: Select Amazon Machine Image

When choosing the instance type, a combination of CPU, memory, storage, and networking capacity should meet the application's needs accordingly. The next figure 4.3 shows that combinations:

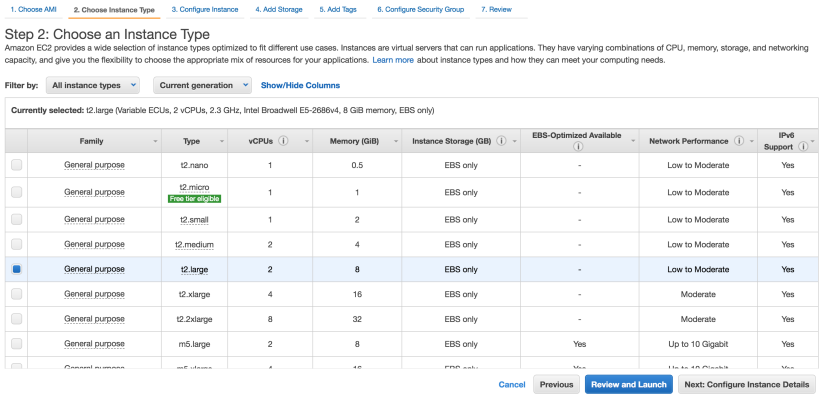


Figure 4.3: Select an Instance Type

Elastic IPs

As it can be observed on the illustration 4.4 below, two instances have been chosen for publishing and authoring purpose. In order to make these instances live and reachable in the Internet, an IP address will be associated by first, allocating it and secondly, associating it to the corresponding machine.

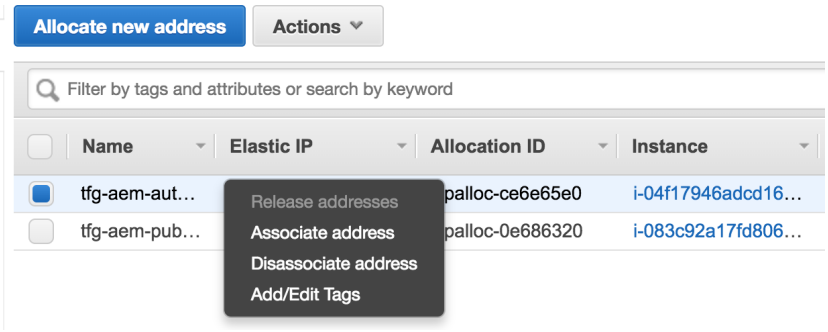


Figure 4.4: Elastic IPs

Installing AEM

Adobe Experience Manager (AEM), is a comprehensive content management solution for building websites, mobile apps and forms. Build lifetime value - deliver digital experiences over the lifetime of your customer that build brand loyalty and drive demand. Get timely and personal - deliver and manage experiences that are responsive, relevant and social. Adobe Experience Manager is an enterprise content management solution that helps simplify the management and delivery of your content and assets. Experience Manager Sites is a content management system within AEM that provides one place to create, manage and deliver digital experiences across websites, mobile sites, and on-site screens. [25]

Since AEM is a java based application, the server should be up to date and have java latest version installed as follows:

```
~$    yum update
~$    yum install java
```

```
~$    scp -i "tfg-aws-keypair.pem" aem-6.3.0.jar
ec2-user@ec2-52-29-87-131.eu-central-1.compute.
    amazonaws.com:.
```

Listing 4.2: Copying AEM to Author

```
~$   scp -i "tfg-aws-keypair.pem" aem-6.3.0.jar
ec2-user@ec2-18-196-127-46.eu-central-1.compute.
   amazonaws.com:.
```

Listing 4.3: Copying AEM to Publish

```
~$   mkdir /opt/cq
~$   mv /home/ec2-user/aem-6.3.0.jar /opt/cq/
~$   cd /opt/cq
~$   java -jar aem-6.3.0.jar -unpack
```

The port and runmode should be adjusted on both Author and Publish instances as follows:

```
11 # TCP port used for stop and status scripts
12 if [ -z "$CQ_PORT" ]; then
13         CQ_PORT=4502
14 fi
...
21 # runmode(s)
22 # will not be used if repository is already present
23 if [ -z "$CQ_RUNMODE" ]; then
24         CQ_RUNMODE='author'
25 fi
```

Listing 4.4: Authoring Instance

```
11 # TCP port used for stop and status scripts
12 if [ -z "$CQ_PORT" ]; then
13         CQ_PORT=4503
14 fi
...
21 # runmode(s)
22 # will not be used if repository is already present
23 if [ -z "$CQ_RUNMODE" ]; then
24         CQ_RUNMODE='publish'
25 fi
```

Listing 4.5: Publishing Instance

After the application has been installed and the ports adjusted on both instances it can be launched by running the following command in the Command Line:

```
~$  ./opt/cq/crx-quickstart/bin/start
```

4.3 Dispatcher

4.3.1 Installing Dispatcher

The procedure to install the Dispatcher module for the web server is following the steps provided by Adobe in Experience Manager Dispatcher User Guide. [26]
In order to obtain the latest Dispatcher installation file for the Linux x86 64bit and web server httpd 2.4 Adobe AEM Cloud offers a platform, named Adobe PackageShare [27], to download the file. According to the guide, the Dispatcher versions are independent of the CMS application and is compatible with AEM used in the project.

```
~$  wget https://www.adobeaemcloud.com/content/
   companies/
public/adobe/dispatcher/dispatcher/_jcr_content/top/
   download_9
/file.res/dispatcher-apache2.4-linux-x86-64-ssl-4.2.3.
   tar.gz
```

```
~$  tar -xzf dispatcher-apache2.4-linux-x86-64-ssl
   -4.2.3.tar.gz
~$  mv dispatcher-apache2.4-4.2.3.so /usr/lib64/httpd/
   modules/
~$  mv conf/* /etc/httpd/conf.modules.d/
~$  ls /usr/lib64/httpd/modules/
~$  mv dispatcher-apache2.4-4.2.3.so mod_dispatcher-
   apache2.4-4.2.3.so
```

For RHEL platforms using 64-bit userspace the default path for apache modules is `/usr/lib64/httpd/modules/`.

4.3.2 Configuring Dispatcher

Initial Setup

Adobe provides a clear guideline [28] on how to configure the dispatcher, available on the product website. In order to start configuring the dispatcher, the first step is to add the following structure to the `httpd.conf`:

```
53 # Example:
54 # LoadModule foo_module modules/mod_foo.so
55 #
56 Include conf.modules.d/*.conf
57 LoadModule mod_dispatcher.so modules/mod_dispatcher
   .so
```

At this point, the apache module is included to the configuration file and enabled. Configure the minimal setting for the dispatcher.

After starting apache, consult the `error_log` of apache and the `dispatcher_log` if apache and dispatcher started correctly. the `error_log` should say something like:

```
[Tue Jun 12 07:35:14.064514 2018] [mpm_prefork:notice]
    [pid 2833] AH00163: Apache/2.4.33 (Amazon)
   Communique/4.2.3 configured -- resuming normal
   operations
```

and the `dispatcher_log` something like:

```
[Fri Jan 19 17:22:16 2001] [I] [19096] Dispatcher
   initialized (build XXXX)
```

As the final step, take a look at the cachedirectory and check if it gets filled. If everything looks fine, you can reduce the loglevel to 0 again.

```
<IfModule disp_apache2.c>

# location of the configuration file. eg: 'conf/
   dispatcher.any'
DispatcherConfig conf/dispatcher.any

# location of the dispatcher log file. eg: 'logs/
   dispatcher.log'
DispatcherLog    logs/dispatcher.log

# log level for the dispatcher log, can be either
   specified
# as a string or an integer (in parentheses)
# error(0): Errors
# warn(1):  Warnings
# info(2):  Infos
# debug(3): Debug
# trace(4): Trace
DispatcherLogLevel warn
```

```
# if turned on, the dispatcher looks like a normal
   module
DispatcherNoServerHeader Off

# if turned on, request to / are not handled by the
   dispatcher
# use the mod_alias then for the correct mapping
DispatcherDeclineRoot Off

# if turned on, the dispatcher uses the URL already
   processed
# by handlers preceeding the dispatcher (i.e.
   mod_rewrite)
# instead of the original one passed to the web server
   .
DispatcherUseProcessedURL Off

# if turned to 1, the dispatcher does not spool an
   error
# response to the client (where the status code is
   greater
# or equal than 400), but passes the status code to
# Apache, which e.g. allows an ErrorDocument directive
# to process such a status code.
#
# Additionally, one can specify the status code ranges
    that should
# be left to web server to handle, e.g.
#
# DispatcherPassError 400-404,501
DispatcherPassError 0

#
# DispatcherKeepAliveTimeout specifies the number of
   seconds a
# connection to a backend should be kept alive. If not
    set or
# set to zero, connections are not kept alive.
#
#DispatcherKeepAliveTimeout 60

</IfModule>
```

Listen to AEM

Making dispatcher listen the same port AEM does:

```
# The load will be balanced among these render
   instances
    /renders
      {
      /rend01
        {
        # Hostname or IP of the render
        /hostname "127.0.0.1"
        # Port of the render
        /port "4503"
        # Connect timeout in milliseconds, 0 to wait
           indefinitely
        # /timeout "0"
        }
      }
```

Client Headers

Configuring main file `dispatcher.any` by forwarding specified request headers to the remote server.

```
/clientheaders {
        "referer"
        "user-agent"
        "authorization"
        "from"
        "content-type"
        "content-length"
        "accept-charset"
        "accept-encoding"
        "accept-language"
        "accept"
        "host"
        "if-match"
        "if-none-match"
        "if-range"
        "if-unmodified-since"
        "max-forwards"
        "proxy-authorization"
        "proxy-connection"
```

```
        "range"
        "cookie"
        "cq-action"
        "cq-handle"
        "handle"
        "action"
        "cqstats"
        "depth"
        "translate"
        "expires"
        "date"
        "dav"
        "ms-author-via"
        "if"
        "lock-token"
        "x-expected-entity-length"
        "destination"
        "x-forwarded-ssl"
        "x-forwarded-proto"
    }
```

Filter Section

The filter section defines the requests that should be handled by the dispatcher.

```
## allow standard resources
    /0001 { /type "allow" /glob "GET /content*" }
    /0002 { /type "allow" /glob "POST /content*" }
    /0003 { /type "allow" /glob "GET /etc/designs*" }
    /0004 { /type "allow" /glob "GET /etc/clientlibs*"
        }
```

```
# Enable specific mime types in non-public content
   directories
    /0041 { /type "allow" /url "*.css"   }  # enable
       css
    /0042 { /type "allow" /url "*.gif"   }  # enable
       gifs
    /0043 { /type "allow" /url "*.ico"   }  # enable
       icos
    /0044 { /type "allow" /url "*.js"    }  # enable
       javascript
    /0045 { /type "allow" /url "*.png"   }  # enable
       png
```

Caching Content

The cache section regulates what responses will be cached and where.

```
/rules {
    /0000 { /glob "*" /type "allow" }
}
```

Cache Invalidation

The invalidate section defines the pages that are "invalidated" after any activation. The page is also flushed once it is modified and consequently removed from the cache.

```
/invalidate {
    /0001 { /type "deny" /glob "*" }

    /0002 { /type "allow" /glob "*.ico" }
    /0003 { /type "allow" /glob "*.png" }
    /0004 { /type "allow" /glob "*.gif" }
    /0005 { /type "allow" /glob "*.jpg" }
    /0006 { /type "allow" /glob "*.jpeg" }
    /0007 { /type "allow" /glob "*.pdf" }
    /0008 { /type "allow" /glob "*.swf" }
    /0009 { /type "allow" /glob "*.mp4" }
    /0010 { /type "allow" /glob "*.m4v" }
    /0011 { /type "allow" /glob "*.ogv" }
    /0012 { /type "allow" /glob "*.webm" }
}
```

The following restricts activation requests to originate from `localhost` only. The allowedClients section restricts the client IP addresses that are allowed to issue activation requests.

```
/allowedClients {
    /0000 {/glob "*" /type "deny" }
    /0001 {/glob "127.0.0.1" /type "allow" }
}
```

Ignore URL Parameters

The ignoreUrlParams section contains query string parameter names that should be ignored when determining whether some request's output can be cached or delivered from cache.

```
/ignoreUrlParams {
    /0001 { /glob "*" /type "deny" }
    /0002 { /glob "m" /type "allow" }
    /0003 { /glob "csref" /type "allow" }
}
```

Grace Period

A grace period defines the number of seconds a stale, auto-invalidated resource may still be served from the cache after the last activation occurring. Auto-invalidated resources are invalidated by any activation, when their path matches the /invalidate section above. This setting can be used in a setup, where a batch of activations would otherwise repeatedly invalidate the entire cache.

```
/gracePeriod "2"
```

Time To Live (TTL)

Enable `TTL` evaluates the response headers from the backend, and if they contain a `Cache-Control` `max-age` or `Expires` date, an auxiliary, empty file next to the cache file is created, with modification time equal to the expiry date. When the cache file is requested past the modification time it is automatically re-requested from the backend.

```
/enableTTL "1"
```

4.4 Varnish

4.4.1 Varnish and Dispatcher

Caching has always been an essential part of AEM, and for this reason Adobe has always recommended dispatcher as an out of the box free caching solution to deliver dynamic content. Besides, it is easy to install being a module of the http server. Nevertheless, dispatcher can be easily replaced by Varnish even

though both work and are implemented in a different way. So as to get a better understanding of the similarities and differences of the mentioned caching tools, the main features will be analyzed accordingly.

Caching Mechanism

The purpose of both caching mechanisms is to enhance the performance of a web application and reduce the load on the server, however the logic behind dispatcher and Varnish is different. As mentioned in the "Dispatcher Overview" section, the dispatcher caches the incoming requests as static documents as they are delivered in the AEM instance and stores them in the filesystem. On the contrary, Varnish caches both requests and responses, not static files, in memory instead of using disk space, thus it offers far more granularity in caching rules.

Caching Rules

Regarding the caching rules, they are relatively straightforward for dispatcher. Plus, it has also built-in rules for invalidation. But these specified rules, being able to only accept or deny different file extensions, are not flexible enough. By comparison, Varnish manages complex rules needed for the site's cache invalidation and for any kind of HTTP request.

HTTP Methods

Speaking about HTTP requests, [...] responses with status codes that are defined as cacheable by default (e.g., 200, 203, 204, 206, 300, 301, 404, 405, 410, 414, and 501 in this specification) can be reused by a cache with heuristic expiration unless otherwise indicated by the method definition or explicit cache controls [RFC7234]; all other status codes are not cacheable by default. [...], according to RFC 7231 in "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content". In case of dispatcher, only 200 status codes can be cached. On the other hand, Varnish allows to cache all methods permitted by the standard, thus reducing load on the machine.

HTTP Headers

This is the main feature where Varnish makes the difference in contrast with the dispatcher. While dispatcher stores html pages not going deeper, Varnish caches HTTP headers for both requests and response, providing a more detailed and granular information. For instance, a specific `Cache-Control` header in the response can prevent caching as follows: `Cache-Control: no-cache`

Query Parameters

First of all, it is worth stressing that if the request contains a query string with parameters or a question mark the dispatcher assumes that the output depends on the query string given and therefore would not cache, instead it would request the document directly from the AEM instance. By contrast, Varnish does offer the possibility of caching these types of URL, and it even goes further, it allows to optimize the URLs to avoid unnecessary cache misses.

One of the examples is the order of the query string parameters. If users put the query-string parameters in a different order, the URL will be technically different with a different hash, and a cache miss as a result. Varnish helps to eliminate that risk by sorting the query-string parameters alphabetically.

Another example of handling URL with Varnish is removing a trailing question mark. Usually, the question mark is used to indicate the start of the URL parameters. If the question mark is for some reason at the end of the URL, there really are no URL parameters. So Varnish would remove the question mark in order to improve the hit rate of the page.

High Load

The dispatcher sends all requests to the backend until the page is stored in the file system. Under high load, it can be observed the issue that too many parallel requests prevent the dispatcher from caching the page at all. As a consequence, during high peaks the page will be delivered without caching. That demonstrates the timeouts when measuring benchmark for dispatcher. On the other hand, Varnish will queue multiple requests to the same resource and therefore dramatically reduce the number of requests to expensive uncached pages.

Flushing and Invalidation

For cache flushing and invalidation, Adode recommends using the dispatcher. It is easy to use and configure, but it is not flexible enough for the cache invalidation techniques needed today, since the invalidation rules are straightforward as well as the caching ones. By comparison, Varnish is also known for customization in flushing and invalidation.

Compatibility

As for compatibility, the dispatcher is an Adobe tool designed with the only purpose to work with AEM and only in a small number of HTTP servers.

However, Varnish, by being a standalone software, can work with any HTTP server that supports reverse proxy functionality.

4.4.2 Installing Varnish

In order to prepare the environment for the software, required dependencies should be installed first:

```
~$  sudo yum install pygpgme yum-utils
```

Secondly, the Varnish rpm package will be downloaded manually. Amazon also makes available Varnish software in its repositories, but it contains an older version of Varnish.

```
~$  wget --content-disposition https://packagecloud.io
   /varnishcache/varnish5/packages/el/6/varnish
   -5.2.1-1.el6.x86_64.rpm/download.rpm
```

Finally, once the package is downloaded it can be installed on the environment:

```
~$  yum install varnish-5.2.1-1.el6.x86_64.rpm
```

After the installation has been successfully completed the version of the package can be verified with the following command:

```
~$   varnishd -V
varnishd (varnish-5.2.1 revision 67e562482)
Copyright (c) 2006 Verdens Gang AS
Copyright (c) 2006-2015 Varnish Software AS
```

There is a default configuration file for Varnish to be modified to add new caching policies based on the application' needs, which is `/etc/varnish/default.vcl`. The first and the most important change to start hitting Varnish cache is to make the software listen port 4503, where AEM is set:

```
backend default {
    .host = "127.0.0.1";
    .port = "4503";
}
```

Every time the VCL configuration file `default.vcl` is modified it should be compiled and verified for syntax errors by running:

```
~$   service varnish reload
```

Apache should be set to port 8080 in the `/etc/httpd/conf/httpd.conf`:

```
41 #Listen 12.34.56.78:80
42 Listen 8080
```

After that, restart apache as well as Varnish:

```
~$  service httpd reload
```

4.4.3 Configuring Varnish

Access Control Lists (ACL)

As the very first step, the application and sensible content should be protected against attacks and leakage. To achieve this, Varnish, similarly to Apache, allows to restrict access to the individuals who should not perform certain actions similar to illustrated below in the source code, which could be accessing or purging content. The first snippet shows how the different ACL groups are created and their purpose.

```
# Access Contol list of IP not granted access
acl unwanted { "216.58.216.0"/24; }

# IPs allowed to perform PURGE action
acl purge {
        "127.0.0.1";
        "192.168.0.0"/24;
}
```

The second snippet of code is illustrating Varnish denying access by IP according to the configured groups and limiting purge action to the IPs mentioned in the purge group.

```
 sub vcl_recv {
    if (client.ip ~ unwanted) {
            return (synth(430, "Access denied"));
    }

    if (req.method == "PURGE"){
             if (!client.ip ~ purge){
                     return (synth(405, "Purging not
                         allowed from "+ client.ip));
             }
             return(purge);
    }
}
```

Blacklists and Whitelists

In terms of security, the following rule is also protecting the content of being accessed with no granted permission. In this case it is filtered by the requested URL.

```
# Denied access to the directories
if (req.url ~ "/system/" ||
        req.url ~ "/crx/" ||
        req.url ~ "/bin/" ||
        req.url ~ "/tmp/" ||
        req.url ~ "/var/" ||
        req.url ~ "/home/" ||
        req.url ~ "/conf/") {
        return (synth(430, "Access denied"));
}
```

Admin Panel

The administrator's panel is also a sensible content. However, although being processed through the ACL, it should not be cached on the client or application side.

```
sub vcl_recv {
    if (!(req.url ~ "/siteadmin" ||
        req.url ~ "/useradmin")) {
        unset req.http.Cookie;
    }
}
```

Session and Login

As for the Authorization and Cookie headers, credentials and session information are considered as personal data related to the unique individual. Consequently, none of these is stored in cache.

```
sub vcl_recv {
    if (req.http.Authorization || req.http.Cookie) {
        return(pass);
    }
}
```

Health Checks

Along with the security the availability of the server should be confirmed from time to time. The following structure assures the backend server is being poked every 5 seconds with 2 seconds of timeout.

```
backend default {
    .host = "127.0.0.1";
    .port = "4503";
    .probe = {
        .request =
                "HEAD / HTTP/1.1"
                "Host: 127.0.0.1"
                "Connection: close"
                "User-Agent: Varnish Healthcheck";
        .timeout = 2s;
        .interval = 5s;
        .initial = 1;
        .window = 8;
        .threshold = 3;
    }
}
```

Handling Arguments

Since the query parameters should go in the established order, Varnish uses a function to normalize the URL if it was introduced incorrectly on the client side. Otherwise, the URL is considered as new and a hash is generated for it.

```
sub vcl_recv {
    # Normalize the query arguments
    set req.url = std.querysort(req.url);

    # Strip any query string from the URL
    set req.url = regsub(req.url, "\?.*", "");

    # Remove anchors
    set req.url = regsub(req.url, "\#.*", "");
}
```

Static Assets

The easiest way of achieving a significant performance improvement is to start caching static files. There are lots of different types and extensions that are not frequently modified. Nevertheless, not all of the static files should be cached due to their size. Therefore, the selected files presented below are the ones to be cached to reduce the number of backend connections.

```
sub vcl_recv {
    # Cache static files with the following extensions
    if (req.url ~ "\.(bmp|css|csv|doc|docx|js|xls|xlsx
        |xml|gif|jpg|jpeg|swf|png|zip|ico|img|wmf|txt|
        woff|woff2|pdf|png|ppt|pptx|rtf|svg|ttf|less|
        odt|otf|)$") {
             unset req.http.Cookie;
             return(hash);
    }
}
```

Media Assets

There is another technique, other than caching, to process the heavy files mentioned earlier. First of all, the audio and video files should be identified and passed to Varnish backend with the following structure:

```
sub vcl_recv {
    if (req.url ~ "\.(mp3|mp4|rar|rpm|tar|tgz|gz|wav|
        zip|bz2|xz|7z|avi|mov|ogm|mpe?g|mka|mkv|webm)$"
        ) {
         unset req.http.Cookie;
         return (hash);
    }
}
```

Varnish will start streaming these media files byte per byte since the moment they are requested. This ensures users a smooth and continuous flow of data, no matter if it is a video streaming or asset download process.

```
sub vcl_backend_response {
    if (bereq.url ~ "\.(mp3|mp4|rar|rpm|tar|tgz|gz|wav
        |zip|bz2|xz|7z|avi|mov|ogm|mpe?g|mka|mkv|webm)$
        ") {
         unset beresp.http.Set-Cookie;
         set beresp.do_stream = true; }
}
```

Error Pages

Even if the backend is down, Varnish will not cache error pages. Instead, a cached content will be served to the client until it is not considered as stale. By applying the following technique Varnish avoids caching most common error pages. This guarantees the user will get fresh content and as soon as the backend is recovered the user will not fall back to the error page again and again.

```
sub vcl_backend_response {
    if (beresp.status == 404 || beresp.status == 500
        || beresp.status == 502 || beresp.status == 503
         || beresp.status == 504) {
            set beresp.ttl = 0s;
         }
}
```

Handling `TTL`, `max-age` and `s-maxage`

There are three items to distinguish regarding timing when it comes to the expiration of the object on client and backend side. In some cases, it is required the object to be cached on the backend longer than on the client side. The problem is that there is only one Cache-Control property and it is intended for the client browser. In order to change the behaviour of the browser cache Varnish should send an appropriate Cache-Control header to tell the browser for how long it should keep the object fresh. `max-age` specifies the maximum amount of time a resource will be considered fresh in the browser, contrary to the s-maxage, which is intended to be used in proxies. The time-to-live from an s-maxage statement is prioritized over a max-age statement. The following is the order Varnish follows to decide the expiration date of the object.

1. First it looks for `TTL`. If it is set, Varnish will use that value
2. Check if `s-maxage` in the `Cache-Control` header is set
3. Check if `max-age` in the `Cache-Control` header is set
4. Check if the `Expires` header is set
5. If none of the list applies, `TTL` will be set to 120s by default

By default, Varnish will cache requests for two minutes (120s) as it is set in the default parameters in `/etc/sysconfig/varnish`. To adjust this time, the declaration in the VCL file can be modified.

```
sub vcl_backend_response {
    set beresp.ttl = 30d;
}
```

The TTL may vary depending on how often the content is updated and on the amount of traffic of the website. According to the code below, Varnish will cache all the content for 30 days and static file for 5 days as TTL is saying. As for the browser cache, it will keep the content in general for 1 hour, except for styling, dynamic files and images will be kept for 3 days. The Expires header is left empty so it is not taken into account for any timing.

```
sub vcl_backend_response {
   # This is how long Varnish will cache content.
    set beresp.ttl = 30d;

    unset beresp.http.expires;

    # Cache static files for 5 days
    if ((bereq.method == "GET" && bereq.url ~ "\.(bmp|
       css|csv|doc|docx|js|xls|xlsx|xml|gif|jpg|jpeg|
       swf|png|zip|ico|img|wmf|txt|woff|woff2|pdf|png|
       ppt|pptx|rtf|svg|ttf|less|odt|otf|)$")) {
        unset beresp.http.Set-Cookie;
        set beresp.ttl = 5d;
    }

    # How long the client browser should keep the item
    set beresp.http.Cache-Control = "max-age = 3600";

    # How long Varnish should keep the item
    set beresp.http.Cache-Control = "s-maxage = 5d";

    # Override browsers to keep styling and dynamics
       for longer
    if (bereq.url ~ ".minify.*\.(css|js).*") {
        set beresp.http.Cache-Control = "max-age=3d";
    }

    if (bereq.url ~ "\.(css|js).*") {
        set beresp.http.Cache-Control = "max-age=3d";
    }

    # Override browsers to cache longer for images
       than for main content
    if (bereq.url ~ "\.(xml|gif|jpg|jpeg|swf|png|zip|
       ico|img|wmf|txt)$") {
        set beresp.http.Cache-Control = "max-age=3d";
    }
}
```

In the example below, when the image is requested in the browser, Varnish sends the request to the backend and there decides on the expiration according to the implemented logic. Varnishlog generated the steps Varnish went through in order to choose the correct expiration date.

```
-   BereqURL       /content/we-retail/us/en/equipment/
    jcr%3acontent/root/responsivegrid/
    category_teaser_883210151.img.jpeg
```

```
-   TTL            VCL 2592000 10 0 1528883249
-   TTL            VCL 432000 10 0 1528883249
-   BerespHeader   cache-control: max-age = 3600
-   BerespUnset    cache-control: max-age = 3600
-   BerespHeader   cache-control: smax-age = 86400
-   BerespUnset    cache-control: smax-age = 86400
-   BerespHeader   cache-control: max-age = 259200
```

Grace and Keep modes

Varnish also allows to adjust headers related to object duration in cache and sometimes deliver stale objects for some amount of time if needed. It can be observed on the illustration below the caching logic and for how long objects are considered fresh or stale.

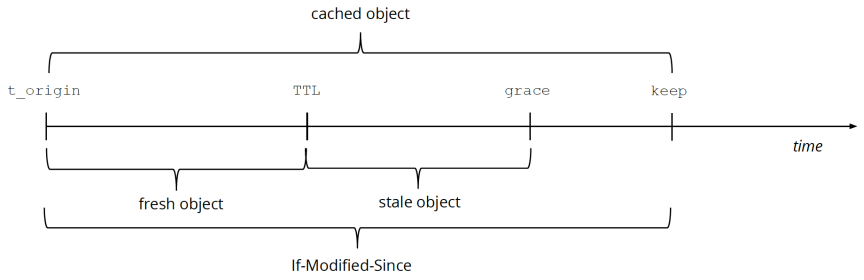


Figure 4.5: Object Lifecycle

At this point, if grace mode is enabled and set to a positive value, Varnish will deliver content to the clients after `TTL` has expired, while it fetches a new version on the background. This can be in the situations when several clients are requesting the same page. Instead of processing multiple requests for the same page, Varnish will send one request to the backend and place the others on hold while fetching one copy from the backend. The grace mode can be adjusted as follows:

```
sub vcl_backend_response {
    # Set Grace time
    set beresp.grace = 1h;

    # Set Keep time
    set beresp.keep = 10m;
}
```

The grace and keep times are also applied to the hit logic in the code below. The first condition checks whether the object is still fresh in cache and deliver it as it is. The second condition verifies if the object has grace or keep time expired, but, still cached. In this case, it will be delivered from cache but fetched again to renew its time to live and make it fresh. If no condition is applied, the object is delivered from the backend and fetched in background for future requests.

```
sub vcl_hit {
    if (obj.ttl >= 0s) {
        # The object is still cached. Serve it as it
            is
        return(deliver);
    }
    if ((obj.ttl + obj.grace > 0s) || (obj.ttl + obj.
        grace + obj.keep > 0s)) {
        # Object is cached becuase of extended time by
            grace and keep
        # Varnish will fetch the object on the
            background
        return(deliver);
    }
    # Else fetch
    return(miss);
}
```

Hit-for-Pass

This feature ensures that high traffic will never overwhelm a backend server. When several users hit the same URL in Varnish, only the first one will be passed on to the backend server, while the rest will wait until the first request comes back from the backend server in the queue. If the content is cacheable, all queued requests will be served the same content, avoiding multiple hits to the backend server. If the the URL is not cacheable, either because the backend sets a cookie, or send `cache-control` private headers, varnish will make what is called a `hit-for-pass` object and cache that for `TTL` or whatever other value

there is set. This `hit-for-pass` object will make sure that the next time users access the URL, varnish already knows that this page is not cacheable, and will send the request to the backend, avoiding the annoying and unfortunate multiple requests.

However, sometimes another scenario occurs, and is that some pages may get stuck with the uncacheable status and time to live assigned. As a consequence, the page will never be served from Varnish, instead, it will always return a MISS and will be served from the backend. That will cause higher latency and response times.

The code below solves this issue by forcing Varnish to generate a hash for the incoming request and deliver the object once it is requested again.

```
sub vcl_hash {
    hash_data(req.url);
    if (req.url ~ "/content/we-retail/|/etc\.|/
        services/") {
        # Workarond for stuck hit-for-pass objects
        hash_data("x");
    }
    if (req.http.host) {
        hash_data(req.http.host);
    } else {
        hash_data(server.ip);
    }
    return (lookup);
 }
```

Hit/Miss Information

When delivering an object, a valuable information with regards to a missed or hit content may be returned to the browser. An example of how to manage to display this information is presented below:

```
sub vcl_deliver {
    # Set how many times the objects has been found in
        cache
    if (obj.hits > 0) {
            set resp.http.X-Cache = "HIT";
            set resp.http.X-Cache-Hits = obj.hits;
    }
    else {
           set resp.http.X-Cache = "MISS";
    }
    set resp.http.WhoisCache = "AEM We.Retail"; }
```

HTTP Method

According to RFC2616, Varnish will only cache resources that are requested through an idempotent HTTP verb, which are HTTP verbs that do not change the state of the resource, which are `GET` and `HEAD`. For the rest of the methods the caching does not make sense since they imply some changes in the request itself.

```
sub vcl_recv {
    if (req.method != "GET" &&
        req.method != "HEAD" &&
        req.method != "PUT" &&
        req.method != "POST" &&
        req.method != "TRACE" &&
        req.method != "OPTIONS" &&
        req.method != "DELETE") {
                /* Non-RFC2616 or CONNECT which is
                   weird. */
                return (pipe);
    }

    if (req.method != "GET" && req.method != "HEAD") {
       /* We only deal with GET and HEAD by default */
       set req.http.X-Pass = "true";
       return (pass);
    }
}
```

Cache Invalidation

Varnish offers a granular way to flush the cache, whether a single file or path is intended to be invalidated or a bunch of them, or the whole cache should be reset. These three options are explained in this section:

1. `Purge` invalidates specific file
2. `Ban` invalidates specific bunch of files
3. `Restart` invalidates entire cache

Purge

The Purge function targets specific object with all its variants from `Vary`, if it was detected for different devices. It also frees up memory and invalidates cache explicitly using objects' hashes. Since the PURGE method can be invoked from the console with curl, as shown on the example further, a verification for the valid purger IP should be done:

```
sub vcl_recv {
    if (req.restarts == 0) {
        unset req.http.X-Purger;
    }

    if (req.method == "PURGE") {
        if (!client.ip ~ purge) {
            return (synth(405, "Purging not allowed
                from " + client.ip));
        }
        return(purge);
    }
}

sub vcl_purge {
    set req.method = "GET";
    set req.http.X-Purger = "Purged";
    return (restart);
}

sub vcl_deliver {
    if (req.http.X-Purger) {
        set resp.http.X-Purger = req.http.X-Purger;
    }
}
```

To invalidate a single object run the following:

```
~$ curl -v -X PURGE http://52.29.87.131:6081/content/
   we-retail/us/en/equipment.html
```

```
* Trying 52.29.87.131...
* TCP_NODELAY set
* Connected to 52.29.87.131 (52.29.87.131) port 6081 (
   #0)
> PURGE /content/we-retail/us/en/equipment.html HTTP
   /1.1
> Host: 52.29.87.131:6081
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 02 Jun 2018 12:40:04 GMT
< X-Content-Type-Options: nosniff
< Content-Type: text/html; charset=UTF-8
< cache-control: max-age = 300
< X-Varnish: 65559
< Age: 0
< Via: 1.1 varnish (Varnish/5.2)
< X-Cache: MISS
< WhoisCache: AEM We.Retail
< X-Purger: Purged
< Accept-Ranges: bytes
< Transfer-Encoding: chunked
< Connection: keep-alive
<
```

Bans

When in need to invalidate more than one item at a time, using bans running from the admin varnish console would be the appropriate option :

```
~$ varnishadm
varnish> ban req.url ~ /content/*
```

Restart

The easiest and quickest way to purge entire cache is simply to just restart Varnish on the server by executing the following command:

```
~$ service varnish restart
```

4.5 Amazon CloudFront CDN

4.5.1 AWS Route53

First thing CloudFront requires, to make the web site public and accessible all over the world, is a valid registered domain name. Route53 is the AWS service that completely fulfills this need and provides multiple choice domain for a variety of prices. As the figure 4.6 shows, Route53 service leads through the process of registration.

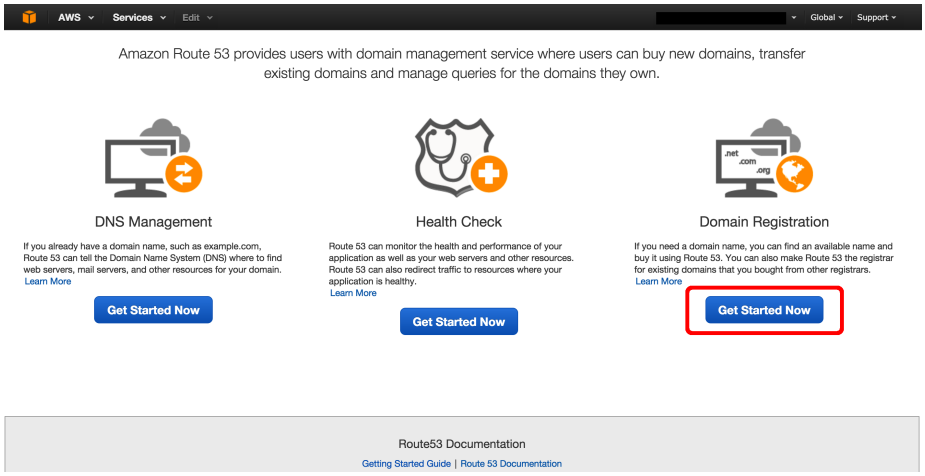


Figure 4.6: Register New Domain

The illustration 4.7 shows the possibility of choosing the name of the developer's choice as well as the domain.

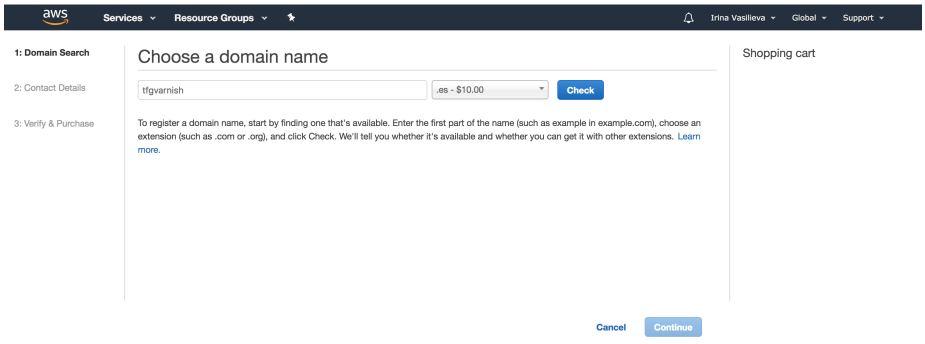


Figure 4.7: Choose Domain Name

As the following figure 4.8 shows, once the domain is registered a public hosted zone is created to route traffic on the internet for a specific domain and its subdomains. In this case, the domain is used to point to the Publish instance, where the dispatcher and varnish are reachable through the corresponding ports.

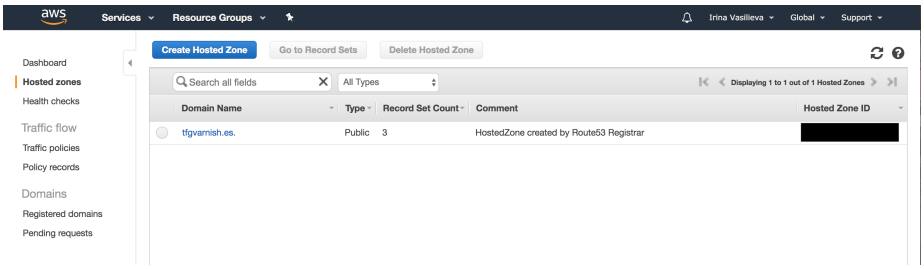


Figure 4.8: Registered Domain

4.5.2 CDN

In order to make the content accessible globally a new distribution should be created in CloudFront. As the figures 4.9 4.10 show, to achieve this goal the Web delivery method should be selected and the newly created distribution assigned to the origin accordingly.

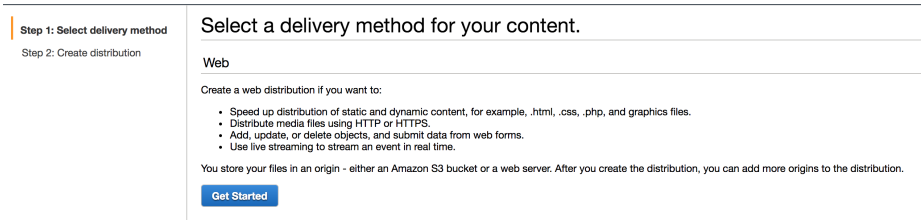


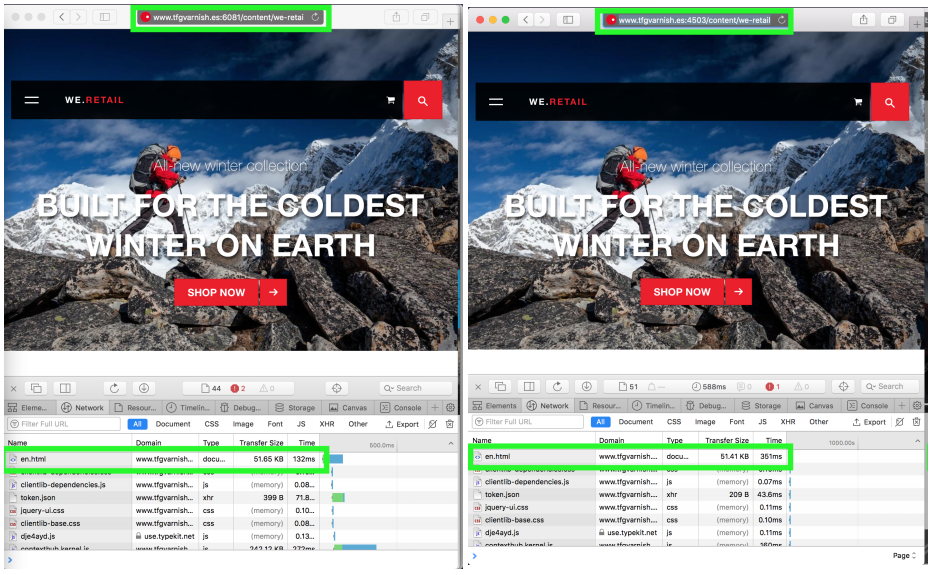
Figure 4.9: Delivery Method

Figure 4.10: Distribution Settings

Chapter 5

Performance Tests

The first validation of the cache improvement can be verified in the browser in the Developer's Tools by checking the response times after the page resulted in a hit.



(a) Varnish Hit

(b) Dispatcher Hit

Figure 5.1: Response Times After Hit

5.1 Varnish Statistics

In order to track the number of hits and miss in Varnish, varnishstat has been started in another console tab. As it can be seen by the uptime of varnish process, Varnish has just been restarted and initialized with no cached content. The number of `MAIN.cache_miss` demonstrates that Varnish sent the request to the backend just once, the rest of the requests were responded from cache, which reach `MAIN.cache_hit` equal 14490 hits.

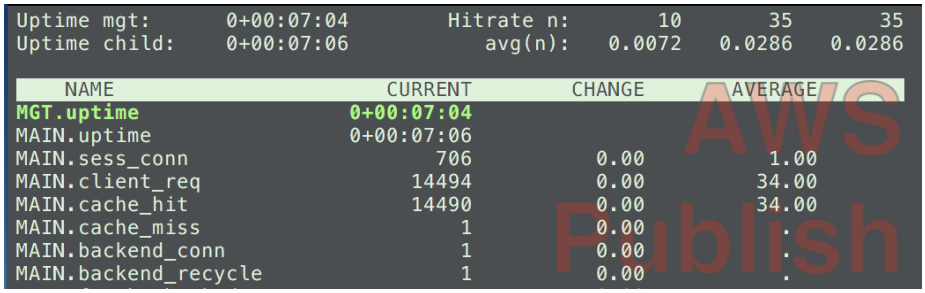


Figure 5.2: Varnish Statistics

5.2 ApacheBench

In order to do the performance tests the limit of simultaneous connections should be taken into consideration for the Dispatcher and Varnish. After having done multiple tests, the final maximum number of connections was defined according to the response capacity and number of time outs given mostly by the dispatcher, as it seems to be less powerful.

4 Types of Tests

T1: 100 requests with 10 concurrent connections
T2: 1000 requests with 100 concurrent connections
T3: 10000 requests with 100 concurrent connections
T4: 10000 requests with 1000 concurrent connections

ApacheBench allows to measure the response times by adjusting the options displayed in the table 5.1, presented further. The tool was running on the local machine, so it could go through all designed layers of the technology stack, by running the following command:

```
ab -g g_results4.tsv -n 10000 -c 1000 -k https://www.
   google.com/
```

-c concurrency
Number of multiple requests to perform at a time. Default is one request at a time.
-n requests
Number of requests to perform for the benchmarking session. The default is to just perform a single request which usually leads to non-representative benchmarking results.
-g gnuplot-file
Write all measured values out as a 'gnuplot' or TSV (Tab separate values) file. This file can easily be imported into packages like Gnuplot, IDL, Mathematica, Igor or even Excel. The labels are on the first line of the file.
-k keep-alive
Enable the HTTP KeepAlive feature, i.e., perform multiple requests within one HTTP session. Default is no KeepAlive.

Table 5.1: Apache HTTP Server Benchmarking Tool Options

Moreover, after all tests with different input have been performed their corresponding results were represented in plot graphics. The requests have been sent to Dispatcher, Varnish and Google to compare their response times. In order to build the plot graphics, a gnuplot has been installed on the local machine by:

```
brew install gnuplot
```

Secondly, the plot script has been adjusted to generate a graphic with the correct labels, size and input. After running the following command the graphic is generated:

```
gnuplot plot.p
```

The corresponding scripts and their results can be found in the Appendix with the following references:

	Test Number	Dispatcher	Varnish	Google
Plot	1	A.1	B.1	C.1
	2	A.2	B.2	C.2
	3	A.3	B.3	C.3
	4	A.4	B.4	C.4
Results	1	A.5	B.5	C.5
	2	A.6	B.6	C.6
	3	A.7	B.7	C.7
	4	A.8	B.8	C.8

Table 5.2: References to Appendices for Plot Scripts and Results

Dispatcher

Test 1

Number of requests	100
Concurrent requests	10

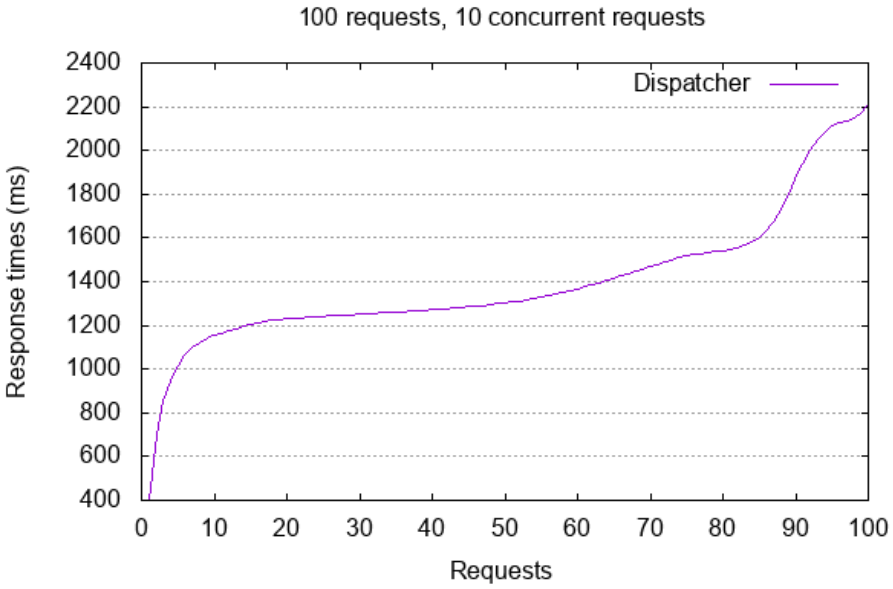


Figure 5.3: 10 Concurrent Requests to Dispatcher

Test 2

Number of requests	1000
Concurrent requests	100

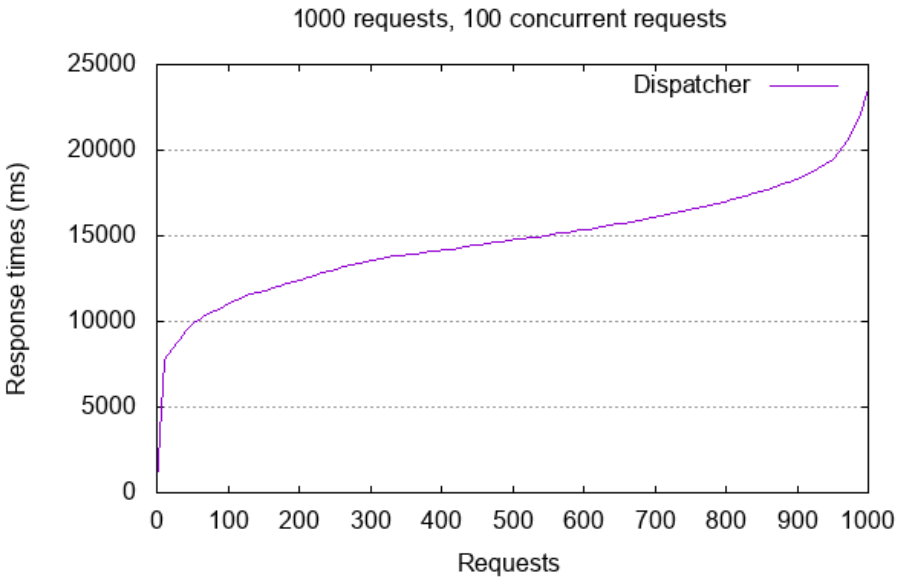


Figure 5.4: 100 Concurrent Requests to Dispatcher

Test 3

Number of requests	10000
Concurrent requests	100

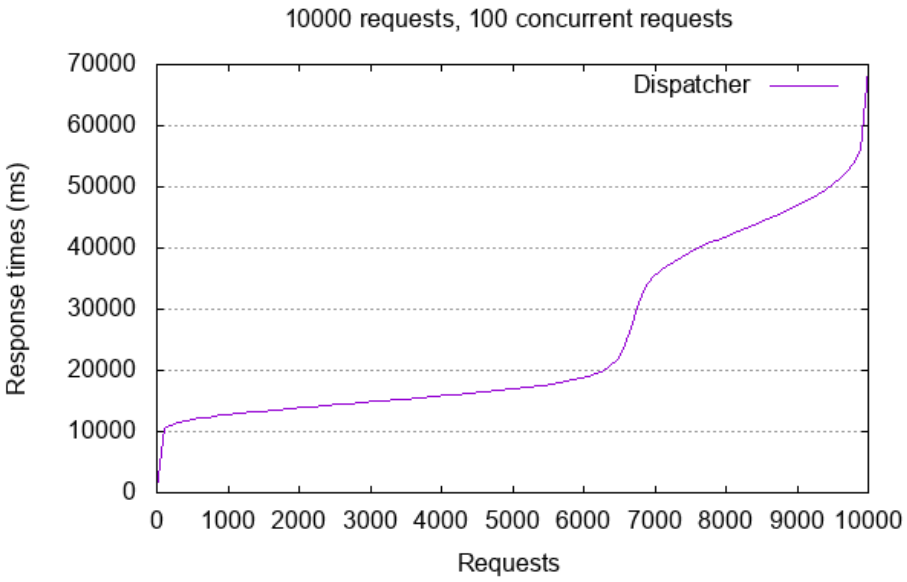


Figure 5.5: 100 Concurrent Requests to Dispatcher

Test 4

Number of requests	10000
Concurrent requests	1000

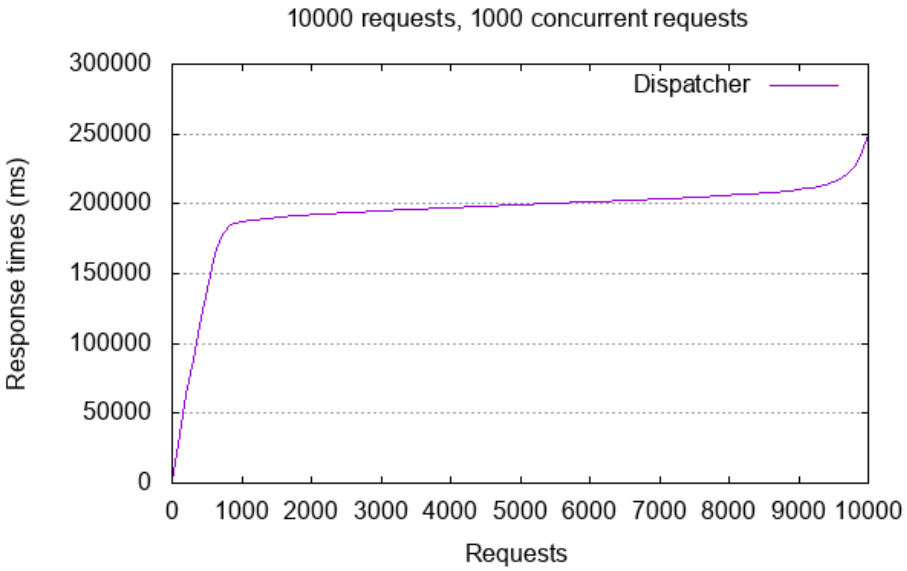


Figure 5.6: 1000 Concurrent Requests to Dispatcher

Varnish

Test 1

Number of requests	100
Concurrent requests	10

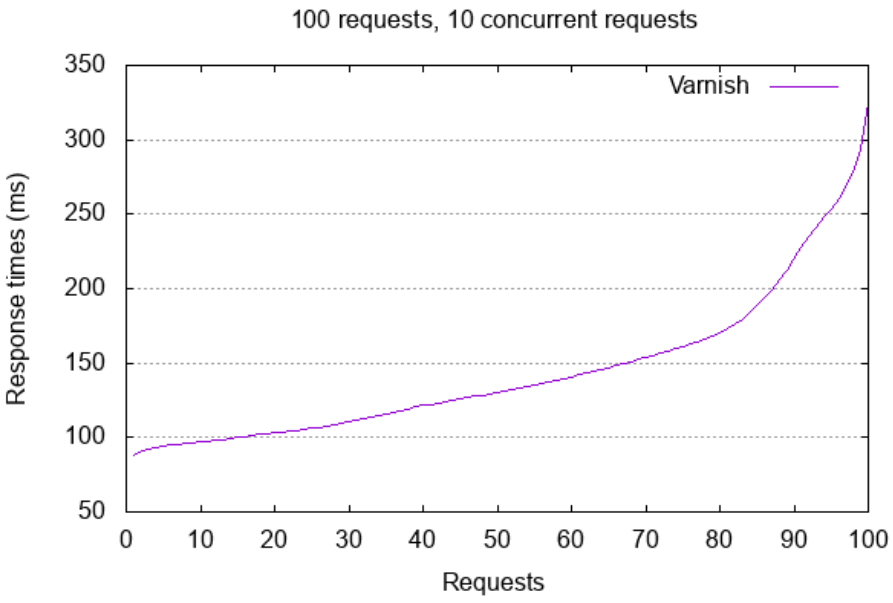


Figure 5.7: 10 Concurrent Requests to Varnish

Test 2

Number of requests	1000
Concurrent requests	100

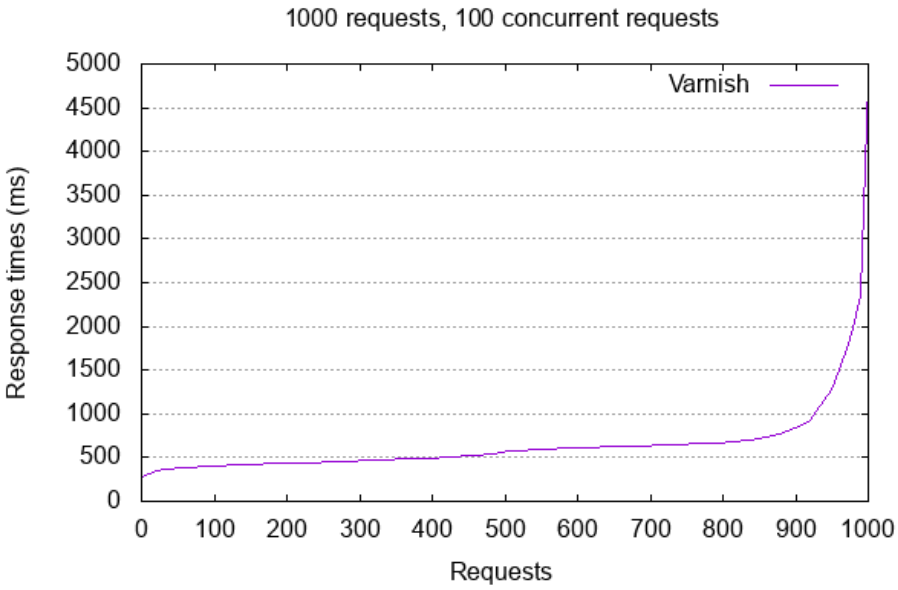


Figure 5.8: 100 Concurrent Requests to Varnish

Test 3

Number of requests	10000
Concurrent requests	100

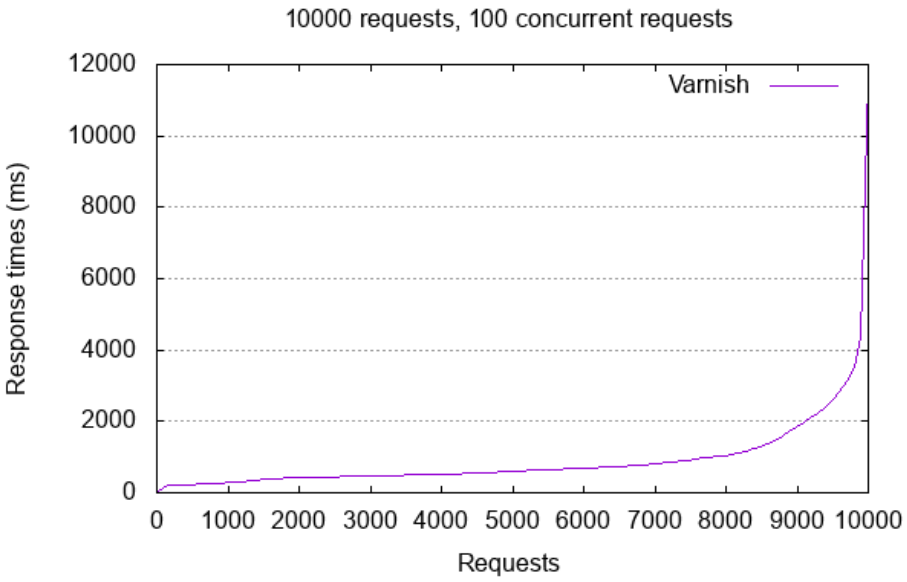


Figure 5.9: 100 Concurrent Requests to Varnish

Test 4

Number of requests	10000
Concurrent requests	1000

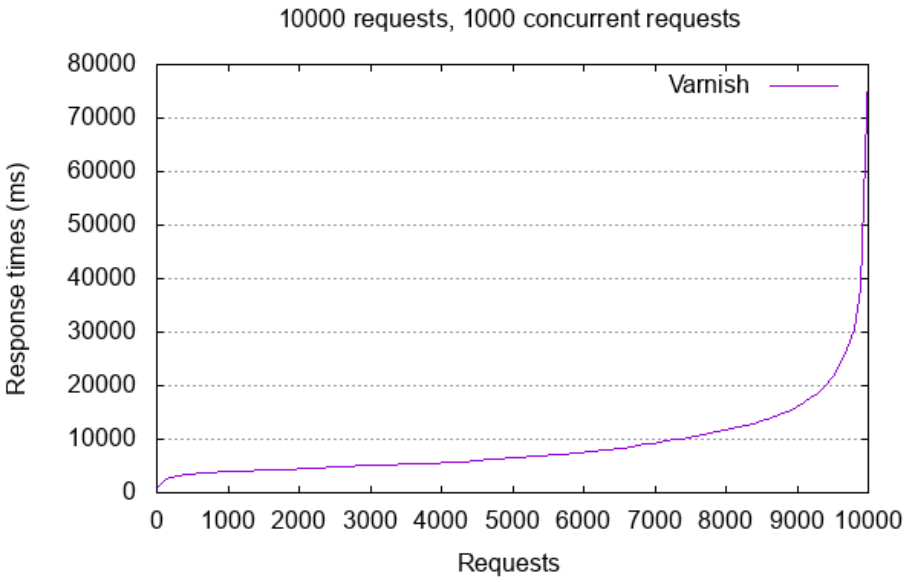


Figure 5.10: 1000 Concurrent Requests to Varnish

Google

Test 1

Number of requests	100
Concurrent requests	10

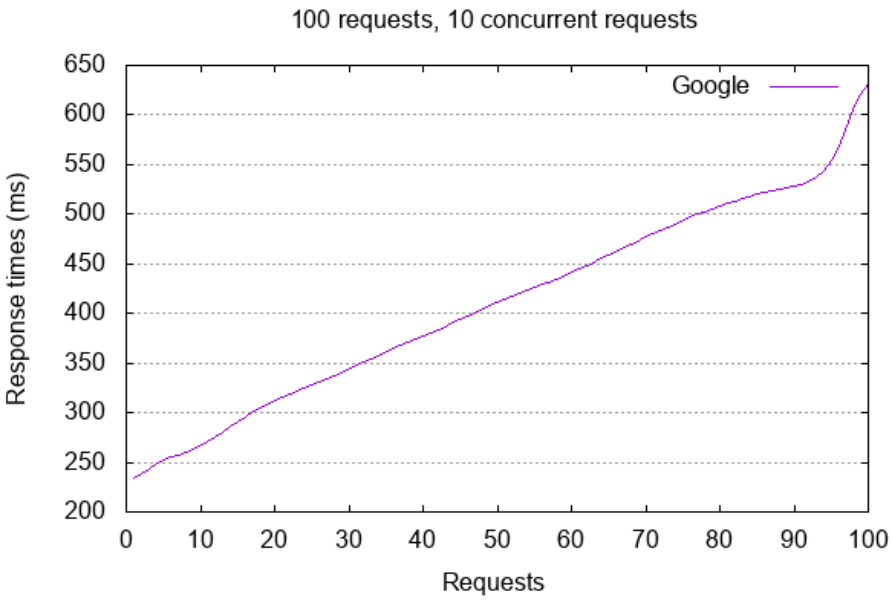


Figure 5.11: 10 Concurrent Requests to Google

Test 2

Number of requests	1000
Concurrent requests	100

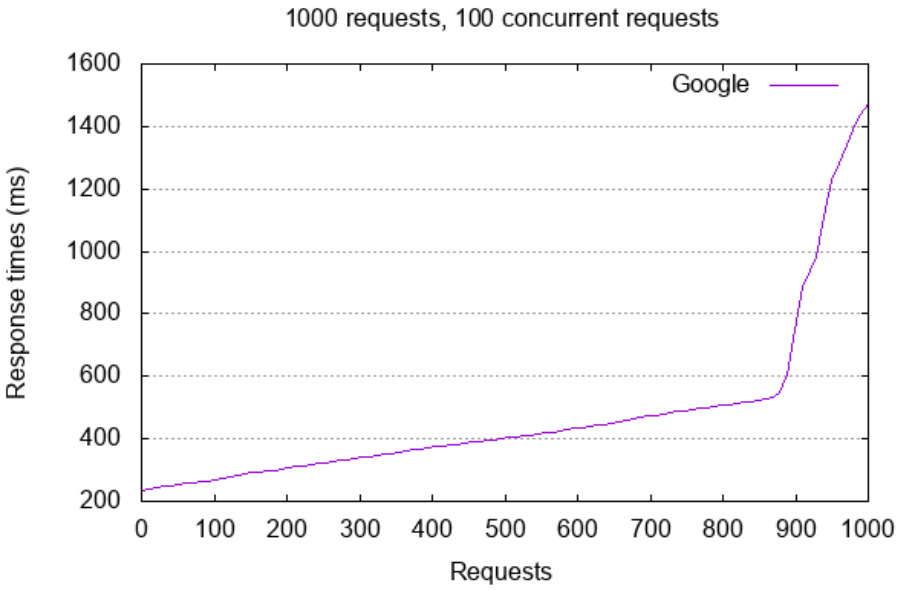


Figure 5.12: 100 Concurrent Requests to Google

Test 3

Number of requests	10000
Concurrent requests	100

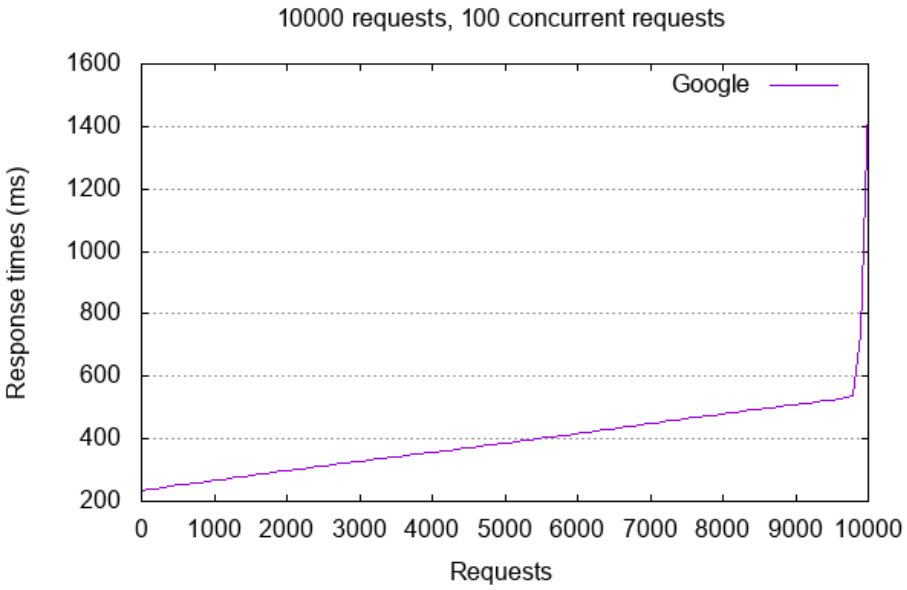


Figure 5.13: 100 Concurrent Requests to Google

Test 4

Number of requests	10000
Concurrent requests	1000

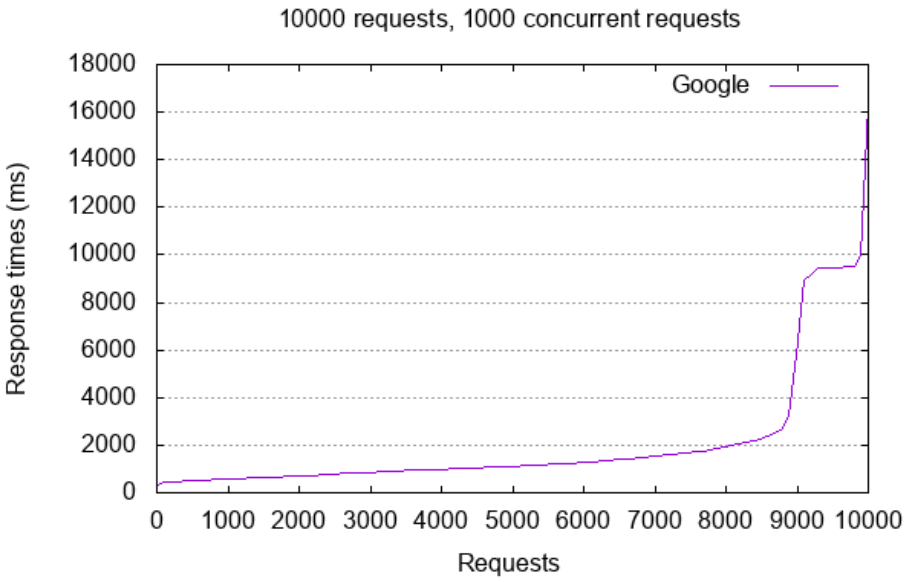


Figure 5.14: 1000 Concurrent Requests to Google

Chapter 6

Budget and Sustainability

6.1 Budget

In the following section different resources are presented in order to prevent unforeseen expenses and calculate an overall price of the project in different aspects such as hardware, software, among others.

6.1.1 Hardware Resources

The are two types of hardware devices that will be used in the project. First of all, a computer with a browser is needed to browse on the Internet, and specially access AWS where the virtual boxes for Author and Publish systems are, which represent the second piece of hardware.

Product	Price	Units	Lifetime	Amortization
MacBookPro 2,8GHz	3.305,59 €	1	10 years	137 €/month
AWS EC2 2.5GHz	0,00 €	2	4 months	0 h/month
Total	-	-	-	548 €/ 4 months

Table 6.1: Hardware Resources

6.1.2 Software Resources

Below all software resources are presented in order to plan and develop the project and, afterwards generate the corresponding documentation when it is complete. To calculate the CloudFront costs during the development of the project an Amazon calculator was used. [29]

Product	Price	Units	Amortization
macOS High Sierra 10.13.3	0,00 €	1	0,00 €
CentOS 7	0,00 €	2	0,00 €
Apache	0,00 €	2	0,00 €
AEM	Trial version	2	0,00 €
Varnish	0,00 €	1	0,00 €
Varnish VMODs	0,00 €	1	0,00 €
Domain	10,00 €	1	10,00 €
Amazon CloudFront	4 €	1	4,00 €
LaTeX	0,00 €	1	0,00 €
Gantt Tomsplanner	0,00 €	1	0,00 €
Vim	0,00 €	1	0,00 €
Total	14,00 €	-	14,00 €

Table 6.2: Software Resources

6.1.3 Human Resources

Below a salary of a system engineer is presented based on the average salaries in this field. [30]

Role	Hours	€/hour	Salary
System Engineer	400	12,00 €/hour	4.800,00 €
Total	400	-	4.800 €

Table 6.3: Human Resources

6.1.4 Indirect Resources

Indirect resources should be taken into account due to value they provide.

Product	Price	Units	Estimated cost
Electricity	0,11281 €/kWh	400 hours	180,00 €
ADSL	40,00 €/month	4 months	160,00 €
Total	-	-	340,00 €

Table 6.4: Indirect Resources

6.1.5 Unforeseen Contingencies

These resources are unforeseen for the developer, but not for the customer, which are the items with a limited trial version and could be ended before the project is presented. It is important to highlight, that the price of AEM figuring in the table is the starting one, and the rest of annual payments is unknown, but usually they are very high.

Product	Price	Units	Time	Total
AEM	365,00 €/license	2	-	730,00 €
Amazon CloudFront	150,00 €/month	1	4 months	600 €
AWS EC2 2.5 GHz Intel Xeon	3,26 €/month	2	4 months	13,04 €
Total	-	-	-	1.343,04 €

Table 6.5: Unforeseen Resources

6.1.6 Total Budget

Finally, the total import of the project is provided below:

Resources	Value
Human	4.800 €
Hardware	548 €
Software	14,00 €
Indirect	340 €
Unforeseen	1.343,04 €
Total	7.045,04 €

Table 6.6: Total Costs

6.2 Sustainability

The project is evaluated in three different aspects with regards to sustainability: economic, environment and social dimensions. On the other hand, these aspects

are also applied to the project in three more different dimensions. All the estimated results are presented in form of calculated values in a table following mentioned analysis.

6.2.1 Environmental Impact Study

No use of a resources was given, but the electricity using the computer to develop the project. Therefore, the only environmental impact is the electricity using directly from the place where the project is developed and indirectly, using AWS servers. However, these companies are already aware of the consequences and have developed solutions around this topic in order to decrease and improve their energy consumption so the impact is not that high. As for AWS,the company is committed to running the business in the most environmentally friendly way possible. [31]

6.2.2 Economic Impact Study

Since this is a bachelor thesis trial versions of some pieces of software are used. But as the main target audience is the enterprises which are already using AEM or willing to switch to AEM, the economy impact should not be as high as for a university student, and it should be no a surprise having to pay certain amount of money for the product or services. Still, there is one item of the technology stack that is completely free being an open source software, and it is Varnish.

Consequently, there could be no savings, nor higher or lower impact for the final customer that would like to set up the technology stack described in the project. And as the infrastructure grows and scales the bills get higher.

6.2.3 Social Impact Study

There is a need for the project due to the fact that Internet technologies and business models are constantly changing and need flexible, forward-compatible technologies that are easy to scale and integrate. Users, in their turn, expect websites and web applications to be fast and reliable, otherwise they will look for other alternatives that are just a mouse-click away and this could lead to a loss of a client for a leading company.

Comparing research data from 2009 with similar research from 2005, Forrester Research measured a significant change in the perception of time. In 2005, online shoppers expected a page to take at least four seconds to load and waited patiently for it to do so. Four years later, in 2009, they expected web pages to load in two seconds or less and would typically abandon a site after three seconds. In a monetary way, nowadays, the speed determines the success of many companies using digital platforms. [8]

	Project Development	**Exploitation**	**Risks**
Environmental	9/10	18/20	0/-20
Economic	8/10	16/20	0/-20
Social	0/10	0/20	0/-20
Total	17/30	34/60	0/-60
Sustainability Range	51/90		

Table 6.7: Sustainability Matrix

Chapter 7

Conclusion

7.1 Acquired Knowledge

The goal of this document was to replace the existing caching tool with an advanced caching layer and proof there was room for improvement. In this project, I have learned more about the existing performance optimization techniques and tools, their benefit to businesses and their impact on users. The core value for every business is to satisfy the customer no matter what. Consequently, enterprises should find their own way to achieve it.

Additionally, I feel more confident working in the designed architecture based on Amazon Web Services. AWS are the most popular products nowadays among enterprise clients. There are a lot of online and offline courses and certifications related to this topic, which highlights the importance and use of the platform. Therefore, this project opened an opportunity to me to dive into a wide range of most-used services to build and deploy web solutions.

Finally, I feel more confident working with Varnish Software. After my first hands-on-approach experience with Varnish, I believe, this is the starting point to get deeper into the advanced learning. In this short period of time, I was able to observe how the behaviour of a website can change by adjusting the appropriate items. I am looking forward to extending my knowledge in other Varnish products as well as in Varnish Cache with the main goal of helping end users to get a better experience, no matter what device is used.

7.2 Project Conclusions

Website optimization is the process of using controlled experimentation to improve a website's ability to drive business goals. Dispatcher is not sufficient to improve the performance of AEM, it needs to be combined with other optimization techniques and tools. On contrary, Varnish is able to compete with Google response times, although it is a standalone server and was being accessed by the testing tool only. On the other hand, Google.com's website doesn't contain the same amount of heavy content, but it is getting a high number of requests per second. Nevertheless, these two facts are directly proportional.

Varnish is a potential software that is able to impress in any environment by pressing the right buttons. It can impact positively the business goals and objectives, providing visitors site the ideal experience and helping them to get a positive impression of the offered products, goods or services. In this quickly highly competitive world on the Internet website performance optimization is always something that should be top priority, where Varnish puts its focus and gets the highest results.

To sum up, in this project Varnish proved it has significant chances to gain user confidence and loyalty and keep business site operational and fast, even with high traffic. Furthermore, running Varnish on a standalone server and delivering high-performance ensures high scalability for any enterprise-sized web application.

List of Figures

List of Tables

Bibliography

[1] Deb Shinder. How to deploy scalable web caching solutions, 2017. URL https://www.techrepublic.com/article/how-to-deploy-scalable-web-caching-solutions/.

[2] Kari Mayhew. What is adobe experience manager (aem)?, 2017. URL http://www.icidigital.com/blog/adobe-experience-manager/what-is-adobe-experience-manager.

[3] AEM. Dispatcher overview, 2017. URL https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher.html.

[4] Thijs Feryn. *Getting started with Varnish Cache*. O'Reilly Media, 2017.

[5] Varnish AG. Case studies, 2017. URL https://www.varnish-software.com/case-studies/.

[6] Datanyze. Top 12 web accelerators, 2017. URL https://www.datanyze.com/market-share/accelerators/Datanyze%20Universe/cloudflare-website-optimization-market-share.

[7] Cloudfare. Cloudflare - web optimization, 2017. URL https://www.cloudflare.com/website-optimization/.

[8] Varnish Software AS. *Whitepaper: The high cost of poor web performance*. Varnish Software, 2013.

[9] Susannah Eriksson. A speedier website at a smaller cost and how wetpaint uses varnish to scale, 2017. URL https://www.redpill-linpro.com/newsletter/whitepaper-varnish-cache.

[10] Vanderbilt University IT. *Configuration Management Process*. Vanderbilt University IT, December 19, 2016.

[11] Jim Murphy Mick MacComascaigh. Magic quadrant for web content management, 26th of July, 2015. URL http://www.awareweb.com/~/media/AwareWeb/Files/WhitePapers/Gartner_Magic_Quadrant_WCM_7-2015.ashx.

[12] Edward Angert. Getting started with varnish cache, February 24, 2017. URL https://linode.com/docs/websites/varnish/getting-started-with-varnish-cache/.

[13] The Apache Software Foundation. ab - apache http server benchmarking tool, 2018. URL https://httpd.apache.org/docs/2.4/programs/ab.html.

[14] Poul-Henning Kamp. Varnish modules, 2018. URL https://varnish-cache.org/vmods/.

[15] AWS. Aws cloud products, 2018. URL https://aws.amazon.com/products/.

[16] AWS. What is amazon ec2?, 2017. URL https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

[17] AWS. Elastic ip addresses, 2018. URL https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html.

[18] AEM. Dispatcher faq, 2017. URL https://helpx.adobe.com/experience-manager/using/dispatcher-faq.html.

[19] Poul-Henning Kamp. Built in subroutines, 2017. URL https://varnish-cache.org/docs/5.2/users-guide/vcl-built-in-subs.html?highlight=subroutines.

[20] Poul-Henning Kamp. Hvarnish processing states, 2017. URL https://varnish-cache.org/docs/trunk/reference/states.html.

[21] Tollef Fog Heen Jérôme Renard Francisco Velázquez, Kristian Lyngstøl. *The Varnish Book*. Redpill Linpro AS, 2018.

[22] Roxana Elliott. The section.io guide to varnish cache, 2017. URL https://www.section.io/blog/varnish-cache-tutorial-vcl/.

[23] Amazon. Amazon cloudfront cdn, 2018. URL https://aws.amazon.com/cloudfront/?nc1=h_ls.

[24] Akamai. Akamai connector for varnish, 2017. URL https://developer.akamai.com/connector-for-varnish/.

[25] Adobe. Make delivering great digital experiences look easy., 2018. URL https://www.adobe.io/apis/experiencecloud/aem.html.

[26] AEM. Installing dispatcher, 2017. URL https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-install.html.

[27] Adobe AEM Cloud. Download dispatcher web server modules, 2017. URL https://www.adobeaemcloud.com/content/companies/public/adobe/dispatcher/dispatcher.html.

[28] AEM. Configuring dispatcher, 2017. URL https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html.

[29] AWS. Simple monthly calculator, 2018. URL http://calculator.s3.amazonaws.com/index.html.

[30] Indeed. System engineer salary, 2017. URL https://www.indeed.es/salaries/Ingeniero/a-en-sistemas-Salaries.

[31] AWS. Aws and sustainability, 2017. URL https://aws.amazon.com/es/about-aws/sustainability/.

Appendices

Appendix A

Dispatcher

A.1 Plot Script 1

```
# Output as png image
set terminal png size 600

# Save file to "results.png"
set output "d_results1.png"

# Graph title
set title "100 requests, 10 concurrent requests"

# Aspect ratio for image size
set size ratio 0.6

# y-axis grid
set grid y

# x-axis label
set xlabel "Requests"

# y-axis label
set ylabel "Response times (ms)"

plot "d_results1.tsv" using 9 smooth sbezier with
   lines title "Dispatcher"
```

Listing A.1: 100 Requests of 10 Concurrent Requests

A.2 Plot Script 2

```
# Output as png image
set terminal png size 600

# Save file to "results.png"
set output "d_results2.png"

# Graph title
set title "1000 requests, 100 concurrent requests"

# Aspect ratio for image size
set size ratio 0.6

# y-axis grid
set grid y

# x-axis label
set xlabel "Requests"

# y-axis label
set ylabel "Response times (ms)"

plot "d_results2.tsv" using 9 smooth sbezier with
   lines title "Dispatcher"
```

Listing A.2: 1000 Requests of 100 Concurrent Requests

A.3 Plot Script 3

```
# Output as png image
set terminal png size 600

# Save file to "results.png"
set output "d_results3.png"

# Graph title
set title "10000 requests, 100 concurrent requests"

# Aspect ratio for image size
set size ratio 0.6

# y-axis grid
```

```
set grid y

# x-axis label
set xlabel "Requests"

# y-axis label
set ylabel "Response times (ms)"

plot "d_results3.tsv" using 9 smooth sbezier with
   lines title "Dispatcher"
```

Listing A.3: 10000 Requests of 100 Concurrent Requests

A.4 Plot Script 4

```
# Output as png image
set terminal png size 600

# Save file to "results.png"
set output "d_results4.png"

# Graph title
set title "10000 requests, 1000 concurrent requests"

# Aspect ratio for image size
set size ratio 0.6

# y-axis grid
set grid y

# x-axis label
set xlabel "Requests"

# y-axis label
set ylabel "Response times (ms)"

plot "d_results4.tsv" using 9 smooth sbezier with
   lines title "Dispatcher"
```

Listing A.4: 10000 Requests of 1000 Concurrent Requests

A.5 Apache Benchmark Test 1

```
ab -g d_results1.tsv -n 100 -c 10 -k http://www.
   tfgvarnish.es:4503/content/we-retail/us/en.html

Server Software:
Server Hostname:        www.tfgvarnish.es
Server Port:            4503

Document Path:          /content/we-retail/us/en.html
Document Length:        52489 bytes

Concurrency Level:      10
Time taken for tests:   14.439 seconds
Complete requests:      100
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      5261800 bytes
HTML transferred:       5248900 bytes
Requests per second:    6.93 [#/sec] (mean)
Time per request:       1443.891 [ms] (mean)
Time per request:       144.389 [ms] (mean, across all
    concurrent requests)
Transfer rate:          355.88 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       38   47   6.8     46      85
Processing:   360 1351 306.6   1262    2162
Waiting:      233 1193 294.1   1108    1959
Total:        401 1398 306.8   1306    2214

Percentage of the requests served within a certain
   time (ms)
  50%   1306
  66%   1428
  75%   1536
  80%   1543
  90%   2092
  95%   2121
  98%   2147
  99%   2214
 100%   2214 (longest request)
```

Listing A.5: 100 Requests of 10 Concurrent Requests

A.6 Apache Benchmark Test 2

```
ab -g d_results2.tsv -n 1000 -c 100 -k http://www.
   tfgvarnish.es:4503/content/we-retail/us/en.html

Server Software:
Server Hostname:        www.tfgvarnish.es
Server Port:            4503

Document Path:          /content/we-retail/us/en.html
Document Length:        52489 bytes

Concurrency Level:      100
Time taken for tests:   151.052 seconds
Complete requests:      1000
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      52618000 bytes
HTML transferred:       52489000 bytes
Requests per second:    6.62 [#/sec] (mean)
Time per request:       15105.187 [ms] (mean)
Time per request:       151.052 [ms] (mean, across all
    concurrent requests)
Transfer rate:          340.18 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd]  median   max
Connect:       37   50  33.4      46      1067
Processing:   428 14703 2916.6  14706   23554
Waiting:      228 14091 2869.9  14108   23151
Total:        476 14753 2914.6  14768   23595

Percentage of the requests served within a certain
   time (ms)
  50%  14768
  66%  15818
  75%  16490
  80%  17055
  90%  18317
  95%  19489
  98%  21503
  99%  22212
 100%  23595 (longest request)
```

Listing A.6: 1000 Requests of 100 Concurrent Requests

A.7 Apache Benchmark Test 3

```
ab -g d_results3.tsv -n 10000 -c 100 -k http://www.
   tfgvarnish.es:4503/content/we-retail/us/en.html


Server Software:
Server Hostname:        www.tfgvarnish.es
Server Port:            4503

Document Path:          /content/we-retail/us/en.html
Document Length:        52489 bytes

Concurrency Level:      100
Time taken for tests:   2490.788 seconds
Complete requests:      10000
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      526180000 bytes
HTML transferred:       524890000 bytes
Requests per second:    4.01 [#/sec] (mean)
Time per request:       24907.880 [ms] (mean)
Time per request:       249.079 [ms] (mean, across all
    concurrent requests)
Transfer rate:          206.30 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd]   median   max
Connect:       37   56   265.6     47    22560
Processing:   781 24800 14011.8  16897   69530
Waiting:      231 23772 13506.7  16206   68496
Total:        859 24856 14020.1  16948   69583

Percentage of the requests served within a certain
   time (ms)
  50%  16948
  66%  24193
  75%  39348
  80%  41944
  90%  47000
  95%  50476
  98%  54044
  99%  55971
 100%  69583 (longest request)
```

Listing A.7: 10000 Requests of 100 Concurrent Requests

A.8 Apache Benchmark Test 4

```
ab -g d_results4.tsv -n 10000 -c 1000 http://www.
   tfgvarnish.es:4503/content/we-retail/us/en.html

Server Software:
Server Hostname:        www.tfgvarnish.es
Server Port:            4503

Document Path:          /content/we-retail/us/en.html
Document Length:        52489 bytes

Concurrency Level:      1000
Time taken for tests:   2012.210 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      526180000 bytes
HTML transferred:       524890000 bytes
Requests per second:    4.97 [#/sec] (mean)
Time per request:       201221.022 [ms] (mean)
Time per request:       201.221 [ms] (mean, across all
    concurrent requests)
Transfer rate:          255.36 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd]   median   max
Connect:       37   76   128.9     46     1101
Processing:  1499 193580 31051.8 199301  250075
Waiting:      666 191961 31119.4 197551  249441
Total:       1541 193656 31016.5 199353  250115

Percentage of the requests served within a certain
   time (ms)
  50%  199353
  66%  202590
  75%  204714
  80%  206230
  90%  210230
  95%  215824
  98%  226962
  99%  234687
 100%  250115 (longest request)
```

Listing A.8: 10000 Requests of 1000 Concurrent Requests

Appendix B

Varnish

B.1 Plot Script 1

```
# Output as png image
set terminal png size 600

# Save file to "results.png"
set output "v_results1.png"

# Graph title
set title "100 requests, 10 concurrent requests"

# Aspect ratio for image size
set size ratio 0.6

# y-axis grid
set grid y

# x-axis label
set xlabel "Requests"

# y-axis label
set ylabel "Response times (ms)"

plot "v_results1.tsv" using 9 smooth sbezier with
   lines title "Varnish"
```

Listing B.1: 100 Requests of 10 Concurrent Requests

B.2 Plot Script 2

```
# Output as png image
set terminal png size 600

# Save file to "results.png"
set output "v_results2.png"

# Graph title
set title "1000 requests, 100 concurrent requests"

# Aspect ratio for image size
set size ratio 0.6

# y-axis grid
set grid y

# x-axis label
set xlabel "Requests"

# y-axis label
set ylabel "Response times (ms)"

plot "v_results2.tsv" using 9 smooth sbezier with
   lines title "Varnish"
```

Listing B.2: 1000 Requests of 100 Concurrent Requests

B.3 Plot Script 3

```
# Output as png image
set terminal png size 600

# Save file to "results.png"
set output "v_results3.png"

# Graph title
set title "10000 requests, 100 concurrent requests"

# Aspect ratio for image size
set size ratio 0.6

# y-axis grid
```

```
set grid y

# x-axis label
set xlabel "Requests"

# y-axis label
set ylabel "Response times (ms)"

plot "v_results3.tsv" using 9 smooth sbezier with
    lines title "Varnish"
```

Listing B.3: 10000 Requests of 100 Concurrent Requests

B.4 Plot Script 4

```
# Output as png image
set terminal png size 600

# Save file to "results.png"
set output "v_results4.png"

# Graph title
set title "10000 requests, 1000 concurrent requests"

# Aspect ratio for image size
set size ratio 0.6

# y-axis grid
set grid y

# x-axis label
set xlabel "Requests"

# y-axis label
set ylabel "Response times (ms)"

plot "v_results4.tsv" using 9 smooth sbezier with
    lines title "Varnish"
```

Listing B.4: 10000 Requests of 1000 Concurrent Requests

B.5 Apache Benchmark Test 1

```
ab -g v_results1.tsv -n 100 -c 10 -k http://www.
   tfgvarnish.es:6081/content/we-retail/us/en.html

Server Hostname:        www.tfgvarnish.es
Server Port:            6081

Document Path:          /content/we-retail/us/en.html
Document Length:        52489 bytes

Concurrency Level:      10
Time taken for tests:   1.561 seconds
Complete requests:      100
Failed requests:        0
Keep-Alive requests:    100
Total transferred:      5288347 bytes
HTML transferred:       5248900 bytes
Requests per second:    64.05 [#/sec] (mean)
Time per request:       156.120 [ms] (mean)
Time per request:       15.612 [ms] (mean, across all
   concurrent requests)
Transfer rate:          3307.97 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    5  14.5      0      55
Processing:    88  139  38.7    129     269
Waiting:       37   49   9.4     48      85
Total:         88  144  49.4    129     324

Percentage of the requests served within a certain
   time (ms)
  50%    129
  66%    150
  75%    162
  80%    169
  90%    222
  95%    252
  98%    277
  99%    324
 100%    324 (longest request)
```

Listing B.5: 100 Requests of 10 Concurrent Requests

B.6 Apache Benchmark Test 2

```
ab -g v_results2.tsv -n 1000 -c 100 -k http://www.
   tfgvarnish.es:6081/content/we-retail/us/en.html

Server Hostname:        www.tfgvarnish.es
Server Port:            6081

Document Path:          /content/we-retail/us/en.html
Document Length:        52489 bytes

Concurrency Level:      100
Time taken for tests:   6.977 seconds
Complete requests:      1000
Failed requests:        0
Keep-Alive requests:    1000
Total transferred:      52883533 bytes
HTML transferred:       52489000 bytes
Requests per second:    143.32 [#/sec] (mean)
Time per request:       697.728 [ms] (mean)
Time per request:       6.977 [ms] (mean, across all
   concurrent requests)
Transfer rate:          7401.74 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    5  16.0      0      64
Processing:   273  630 381.0    563    4775
Waiting:       39  211 109.4    206    1447
Total:        273  636 382.2    567    4833

Percentage of the requests served within a certain
   time (ms)
  50%    567
  66%    629
  75%    655
  80%    673
  90%    840
  95%   1277
  98%   1993
  99%   2522
 100%   4833 (longest request)
```

Listing B.6: 1000 Requests of 100 Concurrent Requests

B.7 Apache Benchmark Test 3

```
ab -g v_results3.tsv -n 10000 -c 100 -k http://www.
   tfgvarnish.es:6081/content/we-retail/us/en.html

Server Hostname:        www.tfgvarnish.es
Server Port:            6081

Document Path:          /content/we-retail/us/en.html
Document Length:        52489 bytes

Concurrency Level:      100
Time taken for tests:   88.032 seconds
Complete requests:      10000
Failed requests:        37
   (Connect: 0, Receive: 0, Length: 37, Exceptions: 0)
Keep-Alive requests:    9963
Total transferred:      526973252 bytes
HTML transferred:       523040717 bytes
Requests per second:    113.60 [#/sec] (mean)
Time per request:       880.318 [ms] (mean)
Time per request:       8.803 [ms] (mean, across all
   concurrent requests)
Transfer rate:          5845.87 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    2  34.1      0    1352
Processing:     0  872 870.8    596   11620
Waiting:        0  302 197.4    247    2756
Total:          0  874 872.6    596   11620

Percentage of the requests served within a certain
   time (ms)
  50%    596
  66%    749
  75%    927
  80%   1053
  90%   1859
  95%   2616
  98%   3540
  99%   4384
 100%  11620 (longest request)
```

Listing B.7: 10000 Requests of 100 Concurrent Requests

B.8 Apache Benchmark Test 4

```
ab -g v_results4.tsv -n 10000 -c 1000 http://www.
   tfgvarnish.es:6081/content/we-retail/us/en.html

Server Hostname:        www.tfgvarnish.es
Server Port:            6081

Document Path:          /content/we-retail/us/en.html
Document Length:        52489 bytes

Concurrency Level:      1000
Time taken for tests:   95.204 seconds
Complete requests:      10000
Failed requests:        68
   (Connect: 0, Receive: 0, Length: 68, Exceptions: 0)
Total transferred:      526576369 bytes
HTML transferred:       522716968 bytes
Requests per second:    105.04 [#/sec] (mean)
Time per request:       9520.409 [ms] (mean)
Time per request:       9.520 [ms] (mean, across all
   concurrent requests)
Transfer rate:          5401.39 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0 1136 1206.8    610    7944
Processing:   692 7709 7077.2   5022   78992
Waiting:        0 1671 1439.5   1320   17416
Total:        982 8845 7120.9   6480   79150

Percentage of the requests served within a certain
   time (ms)
  50%   6480
  66%   8417
  75%  10440
  80%  11810
  90%  16024
  95%  21798
  98%  30951
  99%  38777
 100%  79150 (longest request)
```

Listing B.8: 10000 Requests of 1000 Concurrent Requests

Appendix C

Google

C.1 Plot Script 1

```
# Output as png image
set terminal png size 600

# Save file to "results.png"
set output "g_results1.png"

# Graph title
set title "100 requests, 10 concurrent requests"

# Aspect ratio for image size
set size ratio 0.6

# y-axis grid
set grid y

# x-axis label
set xlabel "Requests"

# y-axis label
set ylabel "Response times (ms)"

plot "g_results1.tsv" using 9 smooth sbezier with
   lines title "Google"
```

Listing C.1: 100 Requests of 10 Concurrent Requests

C.2 Plot Script 2

```
# Output as png image
set terminal png size 600

# Save file to "results.png"
set output "g_results2.png"

# Graph title
set title "1000 requests, 100 concurrent requests"

# Aspect ratio for image size
set size ratio 0.6

# y-axis grid
set grid y

# x-axis label
set xlabel "Requests"

# y-axis label
set ylabel "Response times (ms)"

plot "g_results2.tsv" using 9 smooth sbezier with
   lines title "Google"
```

Listing C.2: 1000 Requests of 100 Concurrent Requests

C.3 Plot Script 3

```
# Output as png image
set terminal png size 600

# Save file to "results.png"
set output "g_results3.png"

# Graph title
set title "10000 requests, 100 concurrent requests"

# Aspect ratio for image size
set size ratio 0.6

# y-axis grid
```

```
set grid y

# x-axis label
set xlabel "Requests"

# y-axis label
set ylabel "Response times (ms)"

plot "g_results3.tsv" using 9 smooth sbezier with
    lines title "Google"
```

Listing C.3: 10000 Requests of 100 Concurrent Requests

C.4 Plot Script 4

```
# Output as png image
set terminal png size 600

# Save file to "results.png"
set output "g_results4.png"

# Graph title
set title "10000 requests, 1000 concurrent requests"

# Aspect ratio for image size
set size ratio 0.6

# y-axis grid
set grid y

# x-axis label
set xlabel "Requests"

# y-axis label
set ylabel "Response times (ms)"

plot "g_results4.tsv" using 9 smooth sbezier with
    lines title "Google"
```

Listing C.4: 10000 Requests of 1000 Concurrent Requests

C.5 Apache Benchmark Test 1

```
ab -g g_results1.tsv -n 100 -c 10 -k https://www.
   google.com/

Server Hostname:        www.google.com
Server Port:            443

Document Path:          /
Document Length:        1704 bytes

Concurrency Level:      10
Time taken for tests:   4.798 seconds
Complete requests:      100
Failed requests:        0
Non-2xx responses:      100
Keep-Alive requests:    100
Total transferred:      198500 bytes
HTML transferred:       170400 bytes
Requests per second:    20.84 [#/sec] (mean)
Time per request:       479.821 [ms] (mean)
Time per request:       47.982 [ms] (mean, across all
   concurrent requests)
Transfer rate:          40.40 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0   13  38.8      0     147
Processing:   234  398  88.4    399     554
Waiting:      233  397  88.5    399     554
Total:        234  410  99.2    418     632

Percentage of the requests served within a certain
   time (ms)
  50%    418
  66%    469
  75%    495
  80%    510
  90%    528
  95%    554
  98%    624
  99%    632
 100%    632 (longest request)
```

Listing C.5: 100 Requests of 10 Concurrent Requests

C.6 Apache Benchmark Test 2

```
ab -g g_results2.tsv -n 1000 -c 100 -k https://www.
   google.com/

Server Hostname:        www.google.com
Server Port:            443

Document Path:          /
Document Length:        1704 bytes

Concurrency Level:      100
Time taken for tests:   5.734 seconds
Complete requests:      1000
Failed requests:        0
Non-2xx responses:      1000
Keep-Alive requests:    1000
Total transferred:      1985000 bytes
HTML transferred:       1704000 bytes
Requests per second:    174.41 [#/sec] (mean)
Time per request:       573.371 [ms] (mean)
Time per request:       5.734 [ms] (mean, across all
   concurrent requests)
Transfer rate:          338.08 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0   73 235.3      0     945
Processing:   235  396  93.0    394     935
Waiting:      234  394  92.5    394     927
Total:        235  469 262.6    403    1474

Percentage of the requests served within a certain
   time (ms)
  50%    403
  66%    455
  75%    492
  80%    507
  90%    935
  95%   1240
  98%   1399
  99%   1442
 100%   1474 (longest request)
```

Listing C.6: 1000 Requests of 100 Concurrent Requests

C.7 Apache Benchmark Test 3

```
ab -g g_results3.tsv -n 10000 -c 100 -k https://www.
   google.com/

Server Hostname:        www.google.com
Server Port:            443

Document Path:          /
Document Length:        327 bytes

Concurrency Level:      100
Time taken for tests:   40.237 seconds
Complete requests:      10000
Failed requests:        7
   (Connect: 0, Receive: 0, Length: 7, Exceptions: 0)
Non-2xx responses:      10000
Keep-Alive requests:    10000
Total transferred:      8667833 bytes
HTML transferred:       3279639 bytes
Requests per second:    248.53 [#/sec] (mean)
Time per request:       402.371 [ms] (mean)
Time per request:       4.024 [ms] (mean, across all
   concurrent requests)
Transfer rate:          210.37 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    8  82.3      0     941
Processing:   233  387  87.9    386     937
Waiting:      233  387  88.0    386     937
Total:        233  395 122.8    387    1484

Percentage of the requests served within a certain
   time (ms)
  50%    387
  66%    437
  75%    464
  80%    479
  90%    510
  95%    526
  98%    537
  99%    931
 100%   1484 (longest request)
```

Listing C.7: 10000 Requests of 100 Concurrent Requests

C.8 Apache Benchmark Test 4

```
ab -g g_results4.tsv -n 10000 -c 1000 -k https://www.
   google.com/

Server Hostname:        www.google.com
Server Port:            443

Document Path:          /
Document Length:        1704 bytes

Concurrency Level:      1000
Time taken for tests:   26.444 seconds
Complete requests:      10000
Failed requests:        11
   (Connect: 0, Receive: 0, Length: 11, Exceptions: 0)
Non-2xx responses:      9990
Keep-Alive requests:    9989
Total transferred:      19829554 bytes
HTML transferred:       17022364 bytes
Requests per second:    378.15 [#/sec] (mean)
Time per request:       2644.434 [ms] (mean)
Time per request:       2.644 [ms] (mean, across all
   concurrent requests)
Transfer rate:          732.29 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0  726 2238.3      0   11188
Processing:   303 1290 798.1   1080   11999
Waiting:        0 1249 722.5   1065    7331
Total:        303 2016 2559.2   1107   16357

Percentage of the requests served within a certain
   time (ms)
  50%   1107
  66%   1416
  75%   1688
  80%   1936
  90%   7238
  95%   9473
  98%   9515
  99%  10001
 100%  16357 (longest request)
```

Listing C.8: 10000 Requests of 1000 Concurrent Requests